5 Using DSP Source Code Interface Software

Overview

DSP Source Code Interface software can be categorized into four general areas:

1. DSP and math functions
2. ISRs (analog I/O drivers) for particular DSP/analog hardware
3. operating mode infrastructure code to support instruments, interactive displays, and certain types of simulation or acceleration of host processing
4. real-time operating system and related utilities

There are two versions of Signalogic DSP Source Code Interface software: source code necessary to provide Hypersignal-Macro series software functionality (to support DSP/analog hardware related functions, displays, and instruments), and source code necessary to provide the same type, as well as code generation functionality, for other Signalogic DSP Software components. For a description of Hypersignal DSP Source Code Interfaces, see section 5.0.2, DSP

Source Code Interfaces for Hypersignal. For a revision history list, see Signalogic DSP Software System Guide section 3.3.4, Revision History for DSP Source Code Interface Software.

NOTE: A signed Non-Disclosure Agreement (NDA) is required prior to obtaining DSP Source Code Interface software for a given DSP chip. Basically, the NDA requires work with the DSP Source Code to be for in-house R&D and product-development purposes, and prohibits using the DSP Source Code within, or to help develop, a product competing with Signalogic DSP Software components. If you would like to see a copy of the NDA, please call and ask for one to be faxed to you.

In order to make effective use of DSP Source Code Interface software, appropriate assemblers/linkers are required. C/C++ compilers are optional, but highly recommended if you plan to modify/add a substantial amount of code (especially for floating-point devices). However, keep in mind that compiler output--even that of the best optimizing compilers--is always at least 3-5x slower than hand-optimized code, which can make the difference between a system operating in real-time or not. Taking compiled code and optimizing only time-critical sections can prove an effective compromise.

In all cases, DSP Source Code Interface software has been implemented and tested with the *DSP device manufacturer's development tools* (i.e. assembler, linker, etc.). The version of tools used for testing is indicated in the individual DSP device family sections listed below. Development tools from other vendors may be compatible, but Signalogic assumes no responsibility if not.

Related Topics

Specific DSP Source Code Interface Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

AT&T WE DSP32 Source Code Interface for Hypersignal

AT&T WE DSP32C Source Code Interface for Hypersignal

AT&T WE DSP3210 Source Code Interface for Hypersignal

Motorola DSP5600x Source Code Interface for Hypersignal

Motorola DSP96002 Source Code Interface for Hypersignal

NEC 77220 Source Code Interface for Hypersignal

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

5.0.1 DSP Source Code Interface Documentation

Many parts of the sections concerning Hypersignal DSP Source Code Interfaces in this chapteronline Help file are more or less reprinted from chapter 4 of the Hypersignal-Macro Series Hardware Reference Guide, but with the addition of information helpful in placing DSP Source Code Interface software in perspective with other Signalogic DSP Software components. Additional information includes noting differences the different versions of DSP Source Code Interfaces, when Hypersignal functionality can affect other components using Hypersignal as a DSP Engine (see User Mode and Engine Mode, in Signalogic DSP Software System Guide section 1.1, Relationship of Signalogic DSP Software Components), and precise locations of subjects in all forms of Signalogic DSP Software documentation.

NOTE: Sections reprinted from Hypersignal-Macro Series Hardware Reference Guide chapter 4 may have material that copyrighted © 1985-1990 by Hyperception, Inc., and/or copyrighted © 1991-1995 by Signalogic, Inc. All rights are reserved by their respective copyright holders.

Also, these sections provide a convenient source of online Help when working with DSP Source Code Interface software (editing, running a software simulator, etc.) under Windows. To quickly access this help when operating DOS or protected-mode DOS development tools (e.g. DSP chip manufacturer development tools), do the following:

1. run the development tool from its Windows icon, or from a DOS prompt started from Program Manager window
2. once the development tool is running, use the Alt-Tab keyboard sequence to switch back to Progam Manager, leaving the development tool software active
3. select this online Help section, and make it full screen
4. now use Alt-Tab to switch back to the development tool; use Alt-Tab again to re-activate the online Help when needed

NOTE: DSP Source Code Interface documentation assumes a knowledge of DSP chip programming and mastery of development tool software, including assembler, linker, compiler, software simulator, downloaders, and debuggers.

General DSP Source Code Interface Topics

DSP Source Code Interface Documentation

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interface Application Examples Using Hypersignal

DSP Source Code Access And Control From Hypersignal

Hypersignal DSP Source Code Notes

Hypersignal DSP Source Code License Agreement

DSP Source Code Interfaces for Hypersignal in Engine Mode

5.0.2 DSP Source Code Interfaces for Hypersignal

DSP Source Code Interface software for Hypersignal includes both library routines and also real-time, interrupt-driven structures that allow foreground and background processing in conjunction with sample interrupts. Initially, the source code supports standard Hypersignal operation, including instrumentation displays, continuous transfer to/from disk, and file-based acceleration, with no programming required. In other words, if you were to assemble and link the source code for a specific DSP chip as it was shipped, without

any changes, you would end up with the distribution object file that is sent with every basic Hypersignal package. However, the source code could then be modified and/or added to in order to enhance the baseline Hypersignal operations or add new functions entirely. Changes can be as simple as interfacing to new analog converters or as complicated as simulating and testing complete algorithms by transferring data between the DSP board and Hypersignal software resources (including hard disk and video display) using pre-defined buffer-switching mechanisms and real-time interrupt structures.

NOTE: The DSP Source Code Interface distribution diskette for a given DSP chip family is not normally included in Hypersignal software shipments, unless purchased additionally or separately.

Related Topics

General DSP Source Code Interface Topics

DSP Source Code Interface Documentation

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interface Application Examples Using Hypersignal

DSP Source Code Access And Control From Hypersignal

Hypersignal DSP Source Code Notes

Hypersignal DSP Source Code License Agreement

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP Source Code Interface Application Examples Using Hypersignal

A basic example of using DSP Source Code Interface software with Hypersignal is to perform some type of custom algorithm or feature extraction. At any time, Hypersignal can be instructed to use a specified executable DSP file(s) as needed, causing user-written

algorithm(s) to execute. This type of modification process basically allows you to concentrate on the problem at hand, since the Hypersignal high-level host software performs housekeeping associated with controlling the DSP board, downloading executable code, and transferring blocks of real-data between host and DSP board memory. Furthermore, if you maintain compatibility with internal Hypersignal data and waveform file formats, then more housekeeping tasks such as file creation, graphical analysis, etc. can be left to Hypersignal.

Following are some more detailed examples of using the DSP Source Code Interface with Hypersignal. Lets suppose that you had one of the DSP boards supported by Hypersignal, but needed to interface the board to a target analog section of your own design. If you had the DSP source code for the appropriate chip family, this would be trivial. First, you would change the interrupt service routine for the Hypersignal function that you wanted to get working. Depending on the chip family there is generally a few different interrupt service routines for each function, organized by board manufacturer. The changes you would make here would be minor: ensuring that the code read your A/D port address instead of the supported board's standard address. Next, you might have to change the system initialization module if the interrupt source/level for your analog hardware is different. Everything else would be left alone: buffering of the data, transfer to the high-level host software, and processing of the data (although you might want to change that, too, at some point). Next, you would assemble the modules changed using the "ASM" batch file provided for each chip family. Finally, you would invoke the "LNK" batch file, and copy the resulting linker output file (".OUT", ".BIN", ".HEX", etc. depending on manufacturer convention) to your working Hypersignal directory. When you next invoked Hypersignal, this file would be ready for downloading and subsequent execution without any further effort.

As another example, suppose you wanted to take a supported board, and working with the board's standard analog I/O, implement all or part of a custom algorithm on the board and stream the real-time results to hard disk for later evaluation and display. By instructing Hypersignal to process 2-channel data, and applying your algorithm to every other sample in the input buffers as they become ready, you can achieve convenient real-time storage to disk, and comparison viewing and evaluation in both time and frequency domain of input data vs. algorithm results.

As a third example, consider using a supported board to simulate an entire project, maintaining just enough of the Hypersignal interface and real-time data transfer mechanism to allow real-time display of data, debug modes, and possible streaming of data to disk. After transferring code to your target system (via EPROM), there can be no doubt as to whether encountered problems are due to the software or the target hardware design--the Hypersignal simulation can always be easily run at any time as a check against the software design.

Or, consider a case where the DSP Source Code Interface is used as the basis for software in a new product or system, with embedded intermediate data "probe" points and real-time transfer mechanisms left installed. By connecting a computer to the product in some way (RS-232 or parallel connection), a laptop or notebook computer can with Hypersignal installed could be brought into the field and used for testing, maintenance and/or data collection at any time.

Other examples include adding DSP source code to support multiprocessing, interconnections between multiple boards, and other peripheral bus standards such as DT-Connect, measuring algorithms or parts of algorithms against a particular real-time sampling rate, evaluating the effects of finite word-length, and more.

Working with DSP Source Code Interface software for Hypersignal is very straightforward. The fact that there exists similar implementations for nine (9) different DSP chip families has led to standardization of library calls and operational procedures, from the function level down to the batch files used to facilitate the code modification process. For example, all DSP chip families have similar routines available for post-FFT processing: log, squareroot, arctan, etc. And, at a lower level, all "LNK.BAT" files produce a map file of the form "XXXLINK.MAP" where XXX is "T30" for Texas Instruments TMS320C3x, "M56" for Motorola DSP5600x, "N20" for NEC 77220, etc. Moreover, the modification and download process is also standardized between DSP families. For example, Hypersignal parses and downloads output files from each DSP manufacturer's linker to the DSP board for subsequent execution, obviating the need for user-written object file downloaders.

Most interrupt service routines fully save and restore machine context to protect anything they might modify, and most routines (such as the "snap-in" real-time filter routines that we have included in the source code library) also save and restore anything they might destroy. In all chip families the Digital Oscilloscope and Spectrum Analyzer

functions provide the best examples of full context-save interrupt service routines combined with substantial foreground processing constructed from many of the routines in the source code library.

General DSP Source Code Interface Topics

DSP Source Code Interface Documentation

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interface Application Examples Using Hypersignal

DSP Source Code Access And Control From Hypersignal

Hypersignal DSP Source Code Notes

Hypersignal DSP Source Code License Agreement

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP Source Code Access And Control From Hypersignal

Hypersignal-Macro series software packages contain capabilities which facilitate access and control of Hypersignal DSP source code containing user-defined modifications or additions, including:

1) The ability to specify different executable DSP files to load and run when a DSP-based function or display is invoked from Hypersignal. In the System Configuration menu, entering a filename, such as "new.out", after the board designator will override the Hypersignal default filename.

2) A macro language function (WriteDSPVal) that allows a macro to cause Hypersignal menu and internal values to be written to arbitrary locations in DSP memory.

3) A macro language function (WriteDSPMem) that allows a macro to cause arbitrary values to be written to arbitrary locations in DSP memory.

For more information about the latter two functions, see Hypersignal-Macro Series Users Manual section 7.10, Macro Commands and Functions.

Note that both the WriteDSPMem and WriteDSPVal functions always make full-word accesses, depending on the natural word-length of the DSP device. This can affect the address value used, if the DSP is capable of accessing data pieces smaller than full-word length. Also, note that both functions are capable of accessing all memory types supported by the DSP chip. For a complete list of internal Hypersignal values that can be written to arbitrary DSP source code memory, see Signalogic DSP Software Reference Guide section 3.3, Hypersignal DSP Source Code Variable List, and for fully detailed information about the syntax and usage of these functions, see Hypersignal-Macro Series Users Manual section 7.10, Macro Commands and Functions.

NOTE: For the most flexibility in manipulating and accessing executing DSP source code, the DSPower-HWLib DLL should be used. See section 5.0.3 below, DSP Source Code Interfaces for Hypersignal in Engine Mode

General DSP Source Code Interface Topics

DSP Source Code Interface Documentation

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interface Application Examples Using Hypersignal

DSP Source Code Access And Control From Hypersignal

Hypersignal DSP Source Code Notes

Hypersignal DSP Source Code License Agreement

DSP Source Code Interfaces for Hypersignal in Engine Mode

Hypersignal DSP Source Code Notes

[@k trace, channel]

Below are some miscellaneous notes to keep in mind when working with Hypersignal DSP Source Code Interface software.

1) There are several references to "trace" and "channel" in DSP source code and source code variable descriptions. Traces are references to waveforms in displays or waveform files, where channels refer to the physical analog input or output connection on the DSP/analog

hardware. A logical-to-physical mapping takes place when channel lists are sent from Hypersignal software to executing DSP code.

Hypersignal DSP Source Code License Agreement

Compliance with the following license agreement is one of the provisions in the NDA agreement mentioned above, a signed copy of which is required before delivery of a DSP Source Code Interface can take place:

Hyperception, Inc. and Signalogic, Inc. reserve all rights to source code supplied with "Hypersignal" series software.

USE OF THIS SOURCE CODE BY YOU ("YOU" MEANS THE COMPANY AND/OR INDIVIDUALS THAT ARE USING THE "HYPERSIGNAL" SOFTWARE) FOR PURPOSES OF RESALE OR PROFIT, OR FOR INCLUSION IN, EITHER WHOLE OR IN PART, ANY OTHER SOFTWARE INTENDED FOR SALE OR PROFIT, IS PROHIBITED.

You are free to copy source code sections to make backup copies and to perform experimentation and testing during algorithm and product development and research. If it is discovered that you, through resale, through lending, or allowing through carelessness or inaction, have enabled another party to obtain possession of a section of the DSP source code marked with a Hyperception or Signalogic copyright, which the other party is including within, or with, a product competitive with the "Hypersignal" series software, then HYPERCEPTION AND SIGNALOGIC ARE PREPARED TO PROSECUTE TO THE FULLEST EXTENT OF THE LAW.

YOU ACKNOWLEDGE THAT YOU HAVE READ THE ABOVE AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT AND THE HYPERSIGNAL DSP SOURCE CODE NON-DISCLOSURE AGREEMENT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF ANY AGREEMENT BETWEEN US, INCLUDING ANY OTHER ORAL OR WRITTEN COMMUNICATION.

General DSP Source Code Interface Topics

DSP Source Code Interface Documentation

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interface Application Examples Using Hypersignal

DSP Source Code Access And Control From Hypersignal

Hypersignal DSP Source Code Notes

Hypersignal DSP Source Code License Agreement

DSP Source Code Interfaces for Hypersignal in Engine Mode

5.0.3 DSP Source Code Interfaces for Hypersignal in Engine Mode

When a Hypersignal-Macro series software package is being used as a DSP Engine by another Signalogic DSP Software component, no inherent type of host PC communication or fixed variable/flag locations are assumed. A Hypersignal DSP Source Code Interface executable file is an option, but is certainly not required. If Hypersignal DSP Source Code *is* used, for example if a built-in DSP source code operating mode such as continuous disk acquisition or digital oscilloscope is desired, then the information in section 5.0.2 above, DSP Source Code Interfaces for Hypersignal, still applies.

However, access to the executable DSP code while it is running is no longer dependent on a built-in Hypersignal instrument or function. For example, DSPower-HWLib offers DSPutMem and DSGetMem calls which can be used to send data back and forth to pre-defined buffers or variable locations in DSP onboard or onchip memory. There are also DSPutHVarMem and DSGetHVarMem calls which can be used to access pre-defined Hypersignal variables, if needed. See chapter 3, Using DSPower-HWLib and DSPower-SPLib Software, for more information.

Furthermore, executable DSP files produced from any DSP source code, whether based on a Hypersignal DSP Source Code Interface or not, can be downloaded and run.

General DSP Source Code Interface Topics

DSP Source Code Interface Documentation

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interface Application Examples Using Hypersignal

DSP Source Code Access And Control From Hypersignal

Hypersignal DSP Source Code Notes

Hypersignal DSP Source Code License Agreement

DSP Source Code Interfaces for Hypersignal in Engine Mode

5.1 Texas Instruments TMS320C25 Source Code Interface

Main Topics:

Related Topics:

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface Modification Notes

5.1.1 Texas Instruments TMS320C25 Source Code Interface for Hypersignal

The Hypersignal TMS320C25 DSP Source Code Interface distribution diskette contains all TMS320C25 source code, including linker control

files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic TMS320C25 source code includes analog I/O interrupt service routines for the LSI/Spectrum, Ariel DSP-16 and PC-C25 (formerly called the DSP-C25), Texas Instruments SWDS, and Burr-Brown PCI-20202C TMS320C25 boards.

As is, the TMS320C25 source code is written in modular form, and supports the basic "modes" of operation below:

TMS320C25 Module	**Hypersignal Functions**	**operating mode**
MODUL0	Analog Conversion	mode = 0
MODUL1	Analog Conversion, Playback from Waveform Display	mode = 1
MODUL2	Digital Oscilloscope	mode = 2
MODUL3	FFT file-based acceleration	mode = 3
MODUL4	iFFT file-based acceleration	mode = 4
MODUL5	Spectrum Analyzer	mode = 5
MODUL6	Stimulus & Response operation	mode = 6
MODUL7	Convolution/Correlation file-based acceleration	mode = 7
MODULn	User-Defined function	mode = n

Related Topics

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface File List for Hypersignal

TMS320C25 Source Code Modification Process for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the TMS320C25 boards and generally match up with the foreground processing modules above. Interrupt service routine module names will have "ISR" in the middle of the filename, after two letters indicating the hardware they support (ISRs are usually hardware-specific), and before a digit indicating the operating mode (see section 5.1.1 above, Texas Instruments TMS320C25 Source Code Interface for Hypersignal) that they support, for example "LSISR0". Different interrupt service routines use different interrupt levels for different boards. Examine the interrupt vector and mask initialization in the "T25DEF" (non-executable definitions) and the "T25BEG" module, depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

Related Topics

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface File List for Hypersignal

TMS320C25 Source Code Modification Process for Hypersignal

TMS320C25 Source Code Interface Host PC Communication

If the user wishes to alter the performance of the code, he/she must keep in mind the necessary links between Hypersignal software and the TMS320C25 code. The two-way communication between the PC and TMS320C25 established in the Hypersignal software must not be changed, as the high-level host software expects to read/write certain

memory locations. These memory locations are initialized by table reads from program memory inside module T25BEG, and are suffixed by 'ADD', as in MODADD, TRGADD, etc. *Do not insert any code of any type before these addresses that would cause them to move*. The TMS320C25 source code has been documented to aid the user in these areas.

For all TMS320C25 boards, you should avoid writing to program memory between 20H and 3FFH, which is typically left open for TMS320C25 debuggers, and 42DH and 4FFH, which is used by Hypersignal to store host-DSP communication variables and structures. Also, note that if Hypersignal DSP TMS320C25 code is allowed to run, it stores local variables in onchip data memory locations 0-127, location 2FF, and 896 through 1023 (filter coefficients and delay elements).

Related Topics

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface File List for Hypersignal

TMS320C25 Source Code Modification Process for Hypersignal

TMS320C25 Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground. Use mode 5 source modules

("MODUL5" and modules it calls), and "ISR5" for one example of a real-time, interrupt-driven structure.

Related Topics

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface File List for Hypersignal

TMS320C25 Source Code Modification Process for Hypersignal

5.1.2 TMS320C25 Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal TMS320C25 DSP Source Code Interface distribution diskette:

T25DEF.ASM	variable & constant definition
T25BEG.ASM	beginning vectors, including ISR vectors, and including initialization, including PC-written variables
LSISR0.ASM	continuous A/D ISR for LSI/Spectrum board
ARISR0.ASM	continuous A/D ISR for Ariel DSP-16 board
SWISR0.ASM	continuous A/D ISR for Texas Instruments SWDS board
AIISR0.ASM	continuous A/D ISR for Ariel PC-C25 (formerly DSP-C25) board (with TLC3204x AIC)
BBISR0.ASM B0ISR0.ASM B2ISR0.ASM	continuous A/D ISRs for Burr-Brown PCI-20202C
LSISR1.ASM	continuous D/A ISR for LSI/Spectrum board

ARISR1.ASM	continuous D/A ISR for Ariel DSP-16 board
SWISR1.ASM	continuous D/A ISR for Texas Instruments SWDS board
AIISR1.ASM	continuous D/A ISR for Ariel PC-C25 (formerly DSP-C25) board (with TLC3204x AIC)
BBISR1.ASM	continuous D/A ISR for Burr-Brown PCI-20202C
ISR2.ASM	Digital Scope ISR for all TMS320C25 boards.
ISR5.ASM	Spectrum Analyzer ISR for all TMS320C25 boards.
ISR6.ASM	Stimulus & Response ISR for all TMS320C25 boards.
MODUL0.ASM	foreground A/D-to-disk processing (mode=0)
BBMOD0.ASM	foreground A/D-to-disk processing (mode=0; Burr-Brown board only)
MODUL1.ASM	foreground D/A-from-disk processing (mode=1)
MODUL2.ASM	foreground Digital Oscilloscope processing (mode=2)
MODUL3.ASM	FFT file-based acceleration processing (mode=3)
MODUL4.ASM	iFFT file-based acceleration processing (mode=4)
MODUL5.ASM	foreground Spectrum Analyzer process (mode=5)
MODUL6.ASM	foreground stimulus & response (mode=6)
MODUL7.ASM	convolution/correlation file-based acceleration processing (mode=7)
ARCTAN.ASM	arctangent routine
CONVLV.ASM	convolution/correlation routine
DIVIDE.ASM	fractional and integer divide routines
EXP2.ASM	exponentiation (base-2) routine (needed for fixed-point transfer function calculation)
FFT.ASM	complex FFT/iFFT routine of variable length N ($N=2^{m}$)
FFTOUT.ASM	post-FFT processing (mag, phase, scaling)

FILTER.ASM	invokes correct filter for current channel
FIR.ASM	FIR filter routine
IIR2.ASM	IIR filter cascade struc 2 (modified direct-form)
IIR3.ASM	IIR filter cascade struc 3 (balanced pole-zero)
LOG2.ASM	log2 approximation routine
MAG.ASM	magnitude routine
MAX.ASM	find maximum routine
MIN.ASM	find minimum routine
MOVE.ASM	move data
PHASE.ASM	4-quadrant phase routine (–π to +π; –pi to +pi, complex input)
SINTAB.ASM	sine value lookup-table for FFT use
SPAOUT.ASM	post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation
SQRT.ASM	squareroot routine
WINDOW.ASM	apply pre-defined window (used with FFTs)
XFER.ASM	complex-ratio routine used for transfer function
ASM.BAT	batch file used to assemble any source module: type "ASM filename" (no extension needed)
LNK.BAT	batch file used to link all source modules: type "LNK"
T25LINK.CMD	link control file (batch file for linker)

The distribution disk does not initially contain all .OBJ files necessary to produce a complete baseline, default executable code module when the T25LINK.CMD control file (above) is invoked with the Texas Instruments "dsplnk" utility. Each .ASM module above must be assembled first.

Related Topics

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface File List for Hypersignal

TMS320C25 Source Code Modification Process for Hypersignal

5.1.3 TMS320C25 Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of TMS32025.OUT before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to TMS320C25 source code, keeping in mind the necessary shared variables between Hypersignal and the TMS320C25. The file T25DEF.ASM is well-documented in this area.

2) Assemble all modules changed and any new modules created (using Texas Instruments "COFF" assembler) to ensure that a new .OBJ (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that can be invoked as:

```
ASM filename <car ret>
```

to assemble source code modules (entering the ".ASM" extension after "filename" is optional). This file executes a

```
dspa -l -v25 %1
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the T25LINK.CMD control file (use an ASCII text-editor to update the T25LINK.CMD file).

3) Re-link all modules using Texas Instruments TMS320C25 "dsplnk" linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .OBJ files. LNK.BAT executes a

```
dsplnk T25LINK.CMD
```

statement, where "T25LINK.CMD" is an ASCII linker-command file that specifies all .OBJ files to be linked by the dsplnk.exe program, as well as specific information about memory addresses for some sections. Note that all executable code that is re-locatable starts at 400H. All code has been tested using version 5.05 of "dspa.exe" and "dsplnk.exe".

4) Copy the resulting "TMS32025.OUT" file to directory with Hypersignal software, run Hypersignal, and invoke the appropriate functions. Hypersignal reads the "COFF" file format directly, downloads all code and data, and begins execution at the reset vector (location zero).

Additional Information

TMS320C25 Source Code Interface Modification Notes

Related Topics

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface File List for Hypersignal

TMS320C25 Source Code Modification Process for Hypersignal

TMS320C25 Source Code Interface Modification Notes

1) If you need to change default buffer addresses, please consult the comments and suggestions in the T25DEF.ASM file.

2) The LNK.BAT file produces a "T25LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

3) The T25LINK.CMD file creates an uninitialized "hole" from 20H through 3FFH that can be used for monitor and debugger programs. Because this section is uninitialized, the Hypersignal COFF loader will not write anything to this area.

4) If you need to use an external sampling clock, follow the basic guidelines below:

1. Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.
2. Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.
3. If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". The value is

in Hz (Hertz); note that for TMS320C25 boards, which use 16-bit wide memory accesses, the maximum value that can be represented is limited to 65535. This value is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).

4. Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface File List for Hypersignal

TMS320C25 Source Code Modification Process for Hypersignal

5.2 AT&T WE DSP32 Source Code Interface

Main Topics:

Related Topics:

AT&T WE DSP32 Source Code Interface for Hypersignal

DSP32 Source Code Interface Modification Notes

5.2.1 AT&T WE DSP32 Source Code Interface for Hypersignal

The DSP32 DSP Source Code Interface distribution diskette contains all AT&T WE DSP32 source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic DSP32 source code includes hardware support code for Communications Automation and Control (CAC) and Burr-Brown ZPB32 WE DSP32 boards.

As is, the DSP32 source code is written in modular form, and supports the basic "modes" of operation below:

DSP32 Module	Hypersignal Functions	operating mode
DSP32.S	Analog Conversion	mode=2
DSP32.S	Analog Conversion, Playback from Waveform Display	mode=1
DSP32.S	Digital Oscilloscope	mode=3
DSP32.S	FFT file-based acceleration	mode=5
DSP32.S	iFFT file-based acceleration	mode=6
DSP32.S	Spectrum Analyzer	mode=4
DSP32.S	User-Defined function	mode=n

Related Topics

AT&T WE DSP32 Source Code Interface for Hypersignal

DSP32 Source Code Interface File List for Hypersignal

DSP32 Source Code Modification Process for Hypersignal

5.2.2 DSP32 Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal DSP32 DSP Source Code Interface distribution disk:

DSP32.S	contains sections for all functions
GETSIM.COM	generates new DSP32.BIN file from DSP3SIM simulator ASCII output file
GETPRG.COM	generates new DSP32.BIN file from DSP32 board program memory (assumes board is DS3EMU emulator-compatible)

Related Topics

AT&T WE DSP32 Source Code Interface for Hypersignal

DSP32 Source Code Interface File List for Hypersignal

DSP32 Source Code Modification Process for Hypersignal

5.2.3 DSP32 Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of DSP32.BIN before running the "LNK" batch file in case your new code doesn't work.

If you wish to alter the performance or function of the code, you must keep in mind the necessary links between the high-level Hypersignal host software and WE DSP32 code. The two-way communication between the PC and DSP32 established in the high-level host Hypersignal software must not be changed, as the high-level software expects to read/write certain memory locations. The WE DSP32 source code has been documented to aid the user in this area.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to DSP32 source code, keeping in mind the necessary links between Hypersignal and the DSP32. All steps below refer to source code as filename.s. If you start with the source code included on DSP Source Code Interface distribution, then filename.s = DSP32.S in the steps below.

The user may find it convenient to create a batch file to execute the following steps 2-6.

2) Re-assemble the source code using the AT&T "make utility" (assembler and linker) by typing:

```
DSP3MAKE -m2 filename.s <car ret>
```

Add the switch "-l" after "-m2" if you desire a listing file output. Add the switch "-s" if simulator testing will be required--but you will need to move the current FFT window data space at on-chip RAM F000H (memory mode 2) because the simulator wants to store breakpoint code at FE00H. The "asm.bat" file does the same thing as the DSP3MAKE command listed above; type:

```
ASM filename <car ret>
```

3) Run the AT&T simulator program to create a file of op-codes. Type:

```
DSP3SIM filename.o <car ret>
```

where filename is the name of the source program.

After the "$im:" prompt, type

```
mem[0:last_byte].b>"DSP32.BYT" <car ret>
```

where "last_byte" is the address of the last byte of program code, or enter a value, for example 0x1000, that ensures all program code is written to disk (The default amount for the source code in its initial form is 0x1000. If you add a significant amount of code, or change the "buf0adr" base address value, you may need to change the amount of simulator data written to the "dsp32.byt" file.)

To exit the simulator, type:

```
quit <car ret>
```

4) Run the GETSIM program; type:

```
GETSIM <car ret>
```

GETSIM expects a file with the name of "dsp32.byt" although this may be changed by altering the source for GETSIM, getsim.pas. Of course, step 3 above must use the new filename as well.

NOTE: Step 4 converts "DSP32.BYT" from an ASCII file to a binary file and saves it under the name "DSP32.BIN". Any method may be used to convert the simulator output to a binary file of consecutive program op-code bytes; the GETSIM program is just one way.

5) Copy the DSP32.BIN file to the directory with Hypersignal software. Hypersignal DSP software will use this file when executing DSP32 functions.

NOTE: The GETPRG program reads program code directly from DSP32 board memory and stores it in the file "DSP32.BIN". The source for GETPRG, "GETPRG.PAS", is included on the Hypersignal DSP32 DSP Source Code Interface diskette and may be modified to change the number of bytes written out, etc. This represents an alternative method to the use of the simulator as described above; for example, the DSP3EMU emulator is used to load the program (created by the DSP3MAKE utility) into DSP32 board memory, then the GETPRG program is run after exiting the emulator, generating a DSP32.BIN file for use by Hypersignal.

Additional Information

DSP32 Source Code Interface Modification Notes

Related Topics

AT&T WE DSP32 Source Code Interface for Hypersignal

DSP32 Source Code Interface File List for Hypersignal

DSP32 Source Code Modification Process for Hypersignal

DSP32 Source Code Interface Modification Notes

1) When you add code to the DSP32.S file, be aware of the "0x1000" boundary. Hypersignal-Macro high-level code assumes that the start of various data buffers is 0x1000, so code that lands beyond this value could be overwritten. You can check the "DSP32.L" list file The "buf0_adr" variable (location 0x1A) can be changed to instruct the high-level software to make more room for program code. However, a better approach, if possible, might be to temporarily de-activate some functions to make room for additional code, so that the amount of data space beyond 0x1000 can be preserved.

2) If you need to use an external sampling clock, follow the basic guidelines below:

1. Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.
2. Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.
3. If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field). Note that a maximum value of 65536 can be represented, because integer values on the DSP32 are 16-bits.
4. Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

AT&T WE DSP32 Source Code Interface for Hypersignal

DSP32 Source Code Interface File List for Hypersignal

DSP32 Source Code Modification Process for Hypersignal

5.3 Motorola DSP5600x Source Code Interface

Main Topics:

Related Topics:

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface Modification Notes

5.3.1 Motorola DSP5600x Source Code Interface for Hypersignal

The Hypersignal DSP5600x DSP Source Code Interface distribution diskette contains all Motorola DSP5600x source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic DSP5600x source code includes analog I/O interrupt service routines for the LSI/Spectrum, Motorola ADS (with external ADC56000 or 56ADC16 analog I/O options), Ariel PC-56 and DSP-56 boards, and the Techron TEF-20 system.

As is, the DSP5600x source code is written in modular form, and supports the basic "modes" of operation below:

DSP5600x Module	Hypersignal Function	operating mode
MODUL0	Analog Conversion	mode=0
MODUL1	Analog Conversion, Playback from Waveform Display	mode=1
MODUL2	Digital Oscilloscope	mode=2
MODUL3	FFT file-based acceleration	mode=3
MODUL4	iFFT file-based acceleration	mode=4
MODUL5	Spectrum Analyzer	mode=5
MODUL6	Stimulus & Response operation	mode=6
MODUL7	Convolution/Correlation file-based acceleration	mode=7

MODULn User-Defined function mode=n

Related Topics

Motorola DSP5600x Source Code Interface for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface File List for Hypersignal

DSP5600x Source Code Modification Process for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the DSP5600x board and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.3.1 above, Motorola DSP5600x Source Code Interface for Hypersignal) that the routine supports, for example "ISR0" for operating mode zero. Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization in the "M56DEF" (non-executable definitions) and "M56BEG" modules, depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

Related Topics

Motorola DSP5600x Source Code Interface for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface File List for Hypersignal

DSP5600x Source Code Modification Process for Hypersignal

DSP5600x Source Code Interface Host PC Communication

If the user wishes to alter or add code functionality, or improve the functionality and/or performance of existing code, he/she must keep in mind the necessary intercommunication variables between Hypersignal software and DSP5600x code. The two-way communication between the host-PC and DSP5600x set up in the Hypersignal software must not be changed, as the software expects to read/write certain memory locations. These memory locations reside at locations "intercom" and "intermod" and are clearly commented inside M56DEF. *Do not insert any code of any type that would change the current addresses of these variables*. If necessary, you may add further communication variables for your own high-level code. The values of ANABFK, WNDBFK, RIBUFK, and MPBUFK (time data buffer, FFT window data buffer, real/imaginary data buffers, & magnitude/phase data buffers, respectively) can be changed if it becomes necessary to alter memory allocation in your DSP5600x code and communicate these changes to the host PC). The DSP5600x source code has been well-documented to aid the user in these areas.

For all DSP5600x boards, you should avoid using the program memory area between 20H and 0FFH, which is used by Hypersignal to implement a host-interface protocol (20H-3FH), store host-DSP communication variables and structures (40H-7FH) and local DSP variables (80H-8FH). Also, note that for some DSP5600x boards, Hypersignal performs a 'pseudo-reset', in which a host interface vector is used to reset the chip, peripherals, and interrupts, and then jump to the actual reset vector at location zero.

Related Topics

Motorola DSP5600x Source Code Interface for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface File List for Hypersignal

DSP5600x Source Code Modification Process for Hypersignal

DSP5600x Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground.

Use mode 5 source modules ("MODUL5" and modules it calls), and "ISR5", for a good example of a real-time, interrupt-driven structure.

Related Topics

Motorola DSP5600x Source Code Interface for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface File List for Hypersignal

DSP5600x Source Code Modification Process for Hypersignal

5.3.2 DSP5600x Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal DSP5600x DSP Source Code Interface distribution diskette:

M56DEF.ASM	variable & constant definition module
M56VEC.ASM	reset and interrupt vector locations, and host interface
M56BEG.ASM	initialization, including variables, operating modes, interrupt vectors, and analog I/O drivers
ISR0.ASM	continuous A/D ISR for all boards
ISR1.ASM	continuous D/A ISR for all boards
ISR2.ASM	Digital Oscilloscope ISR for all boards
ISR5.ASM	Spectrum Analyzer ISR for all boards
ISR6.ASM	Stimulus & Response ISR for all boards
MODUL0.ASM	foreground A/D-to-disk processing (mode=0)
MODUL1.ASM	foreground D/A-from-disk processing (mode=1)
MODUL2.ASM	foreground Digital Oscilloscope processing (mode=2)
MODUL3.ASM	FFT file-based acceleration processing (mode=3)
MODUL4.ASM	iFFT file-based acceleration processing (mode=4)
MODUL5.ASM	foreground Spectrum Analyzer processing (mode=5)
MODUL6.ASM	foreground stimulus & response processing (mode=6)
MODUL7.ASM	convolution/correlation file-based acceleration processing (mode=7)
ARCTAN.ASM	arctangent routine
CONVLV.ASM	convolution/correlation routine
DIVIDE.ASM	fractional and integer divide routines
EXP2.ASM	exponentiation (base-2) routine (needed for fixed-point transfer function calculation)
FFT.ASM	complex FFT/iFFT routine of variable length N ($N=2^{m**m}$)

FILTER.ASM	filter control routine for real-time filtering
FIR.ASM	FIR filter routine
IIR2.ASM	IIR (cascade sections, modified direct form) filter routine
IIR3.ASM	IIR (cascade sections, balanced pole-zero form) filter routine
LOG2.ASM	log2 approximation routine
MAG.ASM	magnitude routine (complex input)
PHASE.ASM	4-quadrant phase routine (–π to +π; –pi to +pi, complex input)
FFTOUT.ASM	post-FFT processing (mag, phase, scaling)
SPAOUT.ASM	post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation
SQRT.ASM	squareroot routine
WINDOW.ASM	window routine: applies PC-downloaded window for Spectrum Analyzer and FFT
XFER.ASM	complex-ratio routine used for transfer function
XMAX.ASM	location of maximum value in X:buffer data
XMIN.ASM	location of minimum value in X:buffer data
YMAX.ASM	location of maximum value in Y:buffer data
XXMOVE.ASM YYMOVE.ASM XYMOVE.ASM YXMOVE.ASM LMOVE.ASM	move routines between X: and Y: memory
IO.EQU	equate file for I/O locations
MSDNHARD.EQU	hardware equate file for TBS board only
M56LINK.CTL	master linker control file
M56MEM.CTL	another linker control file, containing memory and I/O access wait-state value for all boards execpt TBS

M56TBS.CTL	another linker control file, containing memory and I/O access wait-state value for all TBS boards
ASM.BAT	batch file that can be used to assemble individual source code modules (use in form "ASM filename")
LNK.BAT	batch file that performs linking (use in form "LNK")

The distribution disk does not initially contain all .CLN files necessary to produce a complete baseline, default executable code module when the M56LINK.CTL control file (above) is invoked with the Motorola DSPLNK utility. Each .ASM module above must be assembled first.

Related Topics

Motorola DSP5600x Source Code Interface for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface File List for Hypersignal

DSP5600x Source Code Modification Process for Hypersignal

5.3.3 DSP5600x Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of DSP56001.OUT (and DSP56TBS.OUT, if you are using a Turtle Beach Systems board) before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to DSP5600x source code, keeping in mind the necessary links between Hypersignal and the DSP5600x (see discussion in DSP5600x Source Code Interface Host PC Communication, above, and M56DEF.ASM module, which is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .LNK (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that you may use in the form of:

```
ASM filename <car ret>
```

to assemble source code modules (entering an ".ASM" extension after "filename" is optional). This file executes a

```
asm56000 -B -L %1
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the M56LINK.CTL control file (use an ASCII text-editor to update the M56LINK.CTL file).

3) Re-link all DSP5600x modules using the Motorola DSPLNK linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .LNK files. LNK.BAT executes a

```
dsplnk -Z -RM56MEM.CTL -BDSP56001.OUT -MM56LINK.MAP -
FM56LINK.CTL
```

statement, where "M56LINK.CTL" is an ASCII linker-control file that specifies all .LNK files to be linked by the DSPLNK.EXE program. Note that all executable code that is re-locatable starts at after the M56VEC and M56DEF modules, which contain some absolutely located data (which is concerned with host communications). All code has been tested using version 4.1 of Motorola's ASM56000 and DSPLNK.

4) Copy the resulting "DSP56001.OUT" (or DSP56TBS.OUT, if you are using a Turtle Beach Systems board) file to directory with Hypersignal, re-invoke the Hypersignal software, and run the appropriate function(s). Hypersignal reads the ".OUT" file format directly, downloads all code and data, and begins execution at the reset vector (location zero).

5) If you prefer, change the lnk.bat file to produce some other "filename.out" file, instead of using the same name as the Hypersignal default. You can instruct Hypersignal to use non-default filenames in the System Config menu (see DSP Source Code Access And Control From Hypersignal, in section 5.0.2, DSP Source Code Interfaces for Hypersignal, above, for details).

Additional Information:

DSP5600x Source Code Interface Modification Notes

Related Topics

Motorola DSP5600x Source Code Interface for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface File List for Hypersignal

DSP5600x Source Code Modification Process for Hypersignal

DSP5600x Source Code Interface Modification Notes

1) If you need to change default buffer addresses, please consult the comments and suggestions in the M56DEF.ASM file.

2) The LNK.BAT file produces a "M56LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for

individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

3) NOTE: The R7 and M7 registers are dedicated to the Hypersignal-Macro host-interface, and should not be used in user-defined code, for any reason, ever. This may cause a problem if you are using a C-compiler. If you need to free up the n7 register set, we suggest that you expand the fast interrupts used in the Hypersignal-Macro host-interface definition to long interrupts, and save registers used temporarily on the stack. This will result in a performance decrease, but may be necessary in order to take advantage of the development time saved by using a high-level language to program the DSP5600x.

4) If you need to use an external sampling clock, follow the basic guidelines below:

1. Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.
2. Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.
3. If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ fields).
4. Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

Motorola DSP5600x Source Code Interface for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface File List for Hypersignal

DSP5600x Source Code Modification Process for Hypersignal

5.4 AT&T WE DSP32C Source Code Interface

Main Topics:

Related Topics:

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface Real-Time Processing Structure

DSP32C Source Code Interface Modification Notes

5.4.1 AT&T WE DSP32C Source Code Interface for Hypersignal

The Hypersignal DSP32C DSP Source Code Interface distribution diskette contains all AT&T WE DSP32C source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic DSP32C source code includes analog I/O interrupt service routines for the Ariel DSP-32C and PC-32C (with various analog I/O options), AT&T, CAC (both AT-DSP32C and AC5 series), Burr-Brown (Intelligent Instrumentation) ZPB34, MTT Lory DSP4200,

LSI/Spectrum, National Instruments AT-DSP2200, and Signalogic Sig32C series boards.

As is, the DSP32C source code is written in modular form, and supports the basic "modes" of operation below:

DSP32C Module	Hypersignal Function	operating mode
MODUL0	Analog Conversion	mode=0
MODUL1	Analog Conversion, Playback from Waveform Display	mode=1
MODUL2	Digital Oscilloscope	mode=2
MODUL3	FFT file-based acceleration	mode=3
MODUL4	iFFT file-based acceleration	mode=4
MODUL5	Spectrum Analyzer	mode=5
MODUL6	Stimulus & Response operation	mode=6
MODUL7	Convolution/Correlation file-based acceleration	mode=7
MODULn	User-Defined function	mode=n

Related Topics

AT&T WE DSP32C Source Code Interface for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface Real-Time Processing Structure

DSP32C Source Code Interface File List for Hypersignal

DSP32C Source Code Modification Process for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the DSP32C boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.4.1 above, AT&T WE DSP32C Source Code Interface for Hypersignal) that the routine supports, for example "ISR0" for operating mode zero. Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization ("itable" and PCW register) in RESET.S and W32BEG.S to locate the correct interrupt service routine for a particular function.

Related Topics

AT&T WE DSP32C Source Code Interface for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface Real-Time Processing Structure

DSP32C Source Code Interface File List for Hypersignal

DSP32C Source Code Modification Process for Hypersignal

DSP32C Source Code Interface Host PC Communication

If the user wishes to alter or add code functionality, or improve the functionality and/or performance of existing code, he/she must keep in mind the necessary intercommunication variables between Hypersignal software and DSP32C code. The two-way communication between the host-PC and DSP32C device set up in the Hypersignal software must not be changed, as the software expects to read/write certain memory locations. These memory locations reside at locations "intercom" and "intermod" and are clearly commented inside W32DEF. *Do not insert*

any code of any type that would change the current addresses of these variables. If necessary, you may add further communication variables for your own high-level code. The values of ANABFK, WNDBFK, RIBUFK, and MPBUFK (time data buffer, FFT window data buffer, real/imaginary data buffers, & magnitude/phase data buffers, respectively) can be changed if it becomes necessary to alter memory allocation in your DSP32C code and communicate these changes to the host PC). The DSP32C source code has been well-documented to aid the user in these areas.

For all DSP32C boards, you should avoid using the program memory area between 100H and 2FFH, which is used by Hypersignal to store host-DSP interface variables and structures (100H-1FFH) and local DSP variables (200H-3FFH). Also, note that location 0FCH is used to store the page register value referred to above. This value is available for use; you can use it to ensure that your code attempts to execute in a memory area that physically exists on the DSP32C board.

Related Topics

AT&T WE DSP32C Source Code Interface for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface File List for Hypersignal

DSP32C Source Code Modification Process for Hypersignal

DSP32C Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground.

Use mode 5 source modules ("MODUL5" and modules it calls), and "ISR5", for a good example of a real-time, interrupt-driven structure.

Related Topics

AT&T WE DSP32C Source Code Interface for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface Real-Time Processing Structure

DSP32C Source Code Interface File List for Hypersignal

DSP32C Source Code Modification Process for Hypersignal

5.4.2 DSP32C Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal DSP32C DSP Source Code Interface distribution diskette:

RESET.S	reset vector, wait-state, and "page register" initialization
W32DEF.S	variable & constant definition module
W32BEG.S	initialization, including ISR vectors
ISR0.S	continuous A/D ISR for all DSP32C boards
ISR1.S	continuous D/A ISR for all DSP32C boards
ISR2.S	Digital Oscillscope ISR for all DSP32C boards
ISR5.S	Spectrum Analyzer ISR for all DSP32C boards
ISR6.S	stimulus & response ISR for all DSP32C boards
ARI_TX.S	analog I/O output ISR for special-case boards only
MODUL0.S	foreground A/D to disk processing (mode=0)
MODUL1.S	foreground D/A from disk processing (mode=1)
MODUL2.S	foreground Digital Oscilloscope processing (mode=2)
MODUL3.S	FFT file-based acceleration processing (mode=3)
MODUL4.S	iFFT file-based acceleration processing (mode=4)
MODUL5.S	foreground Spectrum Analyzer process (mode=5)

MODUL6.S	foreground stim & response process (mode=6)
MODUL7.S	convolution/correlation file-based acceleration (mode=7)
ARCTAN.S	arctangent routine
CONVLV.S	convolution/correlation routine
DIV.S	divide routine
FFT.S	complex FFT/iFFT routine of variable length N ($N=2^{m}$**m)
FFTOUT.S	post-FFT processing (mag, phase, etc.)
FILTER.S	filter control routine for real-time filtering
FIR.S	FIR filter routine
IIR2.S	IIR (cascade sections, modified direct form) filter routine
IIR3.S	IIR (cascade sections, balanced pole-zero structure) filter routine
LOG10.S	log (base 10) approximation routine
MAXFIND.S	find maximum-in-buffer routine
MINFIND.S	find minimum-in-buffer routine
MAG.S	magnitude routine (complex input)
PHASE.S	4-quadrant phase routine (–π to +π; –pi to +pi, complex input)
SPAOUT.S	post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation
SQRT.S	squareroot routine
WINDOW.S	apply pre-defined window (used with FFTs)
XFER.S	complex-ratio routine used for transfer function
ASM.BAT	batch file used to assemble any source module: type "ASM filename" (no extension needed)
LNK.BAT	batch file used to link all source modules: type "LNK"

W32LINK.CTL link control file (command file for linker)

The distribution disk does not initially contain all .O files necessary to produce a complete baseline, default executable code module when the W32LINK.CTL control file (above) is invoked with the AT&T d3ld utility. Each .S module above must be assembled first.

Related Topics

AT&T WE DSP32C Source Code Interface for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface Real-Time Processing Structure

DSP32C Source Code Interface File List for Hypersignal

DSP32C Source Code Modification Process for Hypersignal

5.4.3 DSP32C Source Code Modification Process

NOTE: Be sure to save a backup copy of DSP32C.OUT before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to DSP32C source code, keeping in mind the necessary shared variable links between Hypersignal and the DSP32C (see discussion in DSP32C Source Code Interface Host PC Communication, above, and W32DEF.S module, which is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .O (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that you may use in the form of:

```
ASM filename <car ret>
```

to assemble source code modules (*do not enter a ".S" extension after "filename"*). This file executes a

```
d3as -Q -l %1.s
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the W32LINK.CTL control file (use an ASCII text-editor to update the W32LINK.CTL file).

3) Re-link all DSP32C modules using the AT&T DL3D linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .O files. LNK.BAT executes the following statements:

```
D3LD -a -m -o DSP32C.OUT W32LINK.CTL > W32LINK.MAP
@COPY W32LINK.MAP+EOF W32LINK.MAP
@D3NM -f DSP32C.OUT >> TEMP
@COPY W32LINK.MAP+TEMP W32LINK.MAP
@DEL TEMP
```

Additional Information:

DSP32C Source Code Interface Modification Notes

Related Topics

AT&T WE DSP32C Source Code Interface for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface Real-Time Processing Structure

DSP32C Source Code Interface File List for Hypersignal

DSP32C Source Code Modification Process for Hypersignal

DSP32C Source Code Interface Modification Notes

1) Hypersignal DSP32C code uses r19 to store a "page register value" (initially written by the high level software to DSP32C onchip RAM address 00FCH) so that Hypersignal DSP32C code can execute on

different DSP32C boards that use different memory modes (which sometimes causes instructions to be in different 64K segments). *You should not change r19 at any time, for any reason*.

2) Hypersignal DSP32C code uses r18 as a system stack pointer for both foreground and interrupt processing code. When storing CAU registers on the "stack", use the following sequence:

```
*r18++ = rN1e
*r18++ = rN2e
        -
        -
        -
*r18++ = rNNe
```

When restoring CAU registers from the stack, use the following sequence:

```
rN1e = r18 - 4
rNNe = *rN1--
        -
        -
        -
rN2e = *rN1--
rN1e = *rN1
r18e = r18 – K              /* K = 4 x (number of registers stored) */
```

There are three remarkable things about this process. First, the stack is always expected to increment and decrement in 4-byte chunks (always save/restore extended registers). Second, the stack is always expected to point to the *next available (free) location*. Third, when restoring, you can't simply do an r18e = r18 – 4 instruction and begin popping rXX = *r18-- instructions, because an interrupt might occur sometime after the r18 – 4 instruction and before you finish popping; the interrupt routine, if it also used r18 as a stack pointer (which all types of routines should be able to do of a legitimate stack pointer), might then clobber a stored register (or more than one) that hadn't been popped yet.

3) Since AT&T dedicates r20-r22 to special purposes, and Hypersignal DSP32C code uses r19 and r18 as system memory-page register and stack pointer, respectively, this means that *r18-r22 are reserved in the Hypersignal DSP32C source code and should not be used arbitrarily for any reason, ever*.

4) Hypersignal DSP32C source code is built with the "ASM" and "LNK" batch files, which invoke the AT&T "d3as" and "d3ld" assembler and linker utilities, respectively. By examining the ASM and LNK batch files, you should be able to construct an equivalent method of building the code using the AT&T "make" utility instead; however, producing a memory map file will be difficult. Obscure DOS commands have been used to create a map file in the LNK.BAT file, since the AT&T utilities will only print map output to the screen. The map file produced has a filename of "W32LINK.MAP"; you can use this file to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

5) Because the different DSP32C boards supported use different memory modes, Hypersignal DSP32C source code does not specify any memory mode of operation. In fact, the "LNK" batch file invocation of the AT&T linker (d3ld) does not specify any memory mode or configuration, causing the linker to assume contiguous memory starting at zero. This is possible because:

The Hypersignal DSP32C source code always assumes at least 800H of read/write memory available at location zero (true for all memory modes). This area is used for the reset vector, storage of the "page register" value, storage of shared variables between host PC and DSP32C, storage of most DSP32C local variables, and system stack.

Use of pc-relative "pccall" and "pcgoto" instructions that interpret procedure entry points and branch destinations as *offsets from the current pc (program counter) value*, rather than absolute constants.

Use of r19 as a pre-defined "page register" for setting the upper 8 bits of registers being used to access local variables and interrupt table vectors not inside the 0-7FFH memory area. See discussion in 1), above.

6) The DSP32C apparently has a bug in the "call(rNN)" instruction in that only the least-significant 16 bits of the current program counter value are stored in rNN before the branch is taken. This is why all Hypersignal DSP32C code "pccall(rNN)" instructions perform a rNNe = rNN | r19 (logical "OR" with r19) as their latent instruction. We highly recommend that any calls you add to the source code do likewise.

7) In addition to setting them directly in the DSP32C source code, IOC register settings can be controlled from the high-level software in order to accommodate non-standard analog I/O connected to DSP32C serial ports. Inside the Hypersignal-Macro configuration file there is a 32-bit integer IOC value for every DSP32C board currently supported by Hypersignal-Macro--see the "CONFIG.DOC" file on Hypersignal distribution diskette #2 for a list of the sequence of variables in the configuration file. Depending on the board currently in use, one of these values is sent down to the DSP32C board whenever a Hypersignal-Macro real-time function requiring analog I/O is invoked. The corresponding DSP source code variable is "IOC_reg", which is normally used directly to set the IOC register (unless changed by the user in the DSP source code).

8) If you need to use an external sampling clock, follow the basic guidelines below:

1. Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.
2. Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.
3. If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software,

based on what is entered in the current menu (SAMPLING FREQ field).

4. Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

AT&T WE DSP32C Source Code Interface for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface Real-Time Processing Structure

DSP32C Source Code Interface File List for Hypersignal

DSP32C Source Code Modification Process for Hypersignal

5.5 Texas Instruments TMS320C3x Source Code Interface

Main Topics:

Related Topics:

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface Modification Notes

5.5.1 Texas Instruments TMS320C3x Source Code Interface for Hypersignal

The Hypersignal TMS320C3x DSP Source Code Interface distribution diskette contains all Texas Instruments TMS320C3x source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic TMS320C3x source code includes analog I/O interrupt service routines for the LSI/Spectrum system, ASPI Banshee, Sonitech Spirit-30, DSP Research TIGER 30, MTT Lory DSP4100, Domain Technologies DSPCard, Innovative Integration PC31, and Texas Instruments EVM boards.

As is, the TMS320C3x source code includes both library routines and also real-time interrupt driven "modes" of operation that use foreground processing and interrupt service routine processing to implement standard Hypersignal functions. These functions have been written in modular form, and support the basic "modes" of operation below:

TMS320C3x Module	Hypersignal Function	operating mode
MODUL0	Analog Conversion	mode=0
MODUL1	Analog Conversion, Playback from Waveform Display	mode=1
MODUL2	Digital Oscilloscope	mode=2
MODUL3	FFT file-based acceleration	mode=3
MODUL4	iFFT file-based acceleration	mode=4
MODUL5	Spectrum Analyzer	mode=5
MODUL6	Stimulus & Response operation	mode=6
MODUL7	Convolution/Correlation file-based acceleration	mode=7
MODULn	User-Defined function	mode=n

Related Topics

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface File List for Hypersignal

TMS320C3x Source Code Modification Process for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the TMS320C3x boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.5.1 above, TMS320C3x Source Code Interface for Hypersignal) that they support, for example "ISR0". Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization in the initial setup code in "T30BEG", depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

Related Topics

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface File List for Hypersignal

TMS320C3x Source Code Modification Process for Hypersignal

TMS320C3x Source Code Interface Host PC Communication

If the user wishes to alter the performance of the code, he/she must keep in mind the necessary links between Hypersignal software and the TMS320C3x code. The two-way communication between the PC and TMS320C3x processor established in the Hypersignal software must not be changed, as the high-level software expects to read/write certain memory locations. These memory locations are initialized by the high-level software before any code executes on the DSP board (some locations are actually read by the high-level software as well, allowing you to change such "constants" as buffer base addresses). *Do not insert any code of any type before these addresses.* The TMS320C3x source code has been documented to aid the user in these areas.

For all TMS320C3x boards, you should avoid using the program memory areas between 0CH and 0C0H, which is reserved by Texas Instruments, between 0C0H and 400H, which is often used for debuggers and monitors (for Texas Instruments EVM boards, Hypersignal uses 300H to 3FFH for a host interace), between 400H and 47FH, which is used by Hypersignal to store host-DSP interface variables and structures, and between 480H and 4FFH, which is used by Hypersignal TMS320C3x DSP code to store local variables.

Related Topics

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface File List for Hypersignal

TMS320C3x Source Code Modification Process for Hypersignal

TMS320C3x Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground. The dual-buffer communication between the host and DSP is fully synchronized using "interlock" buffer flags. Use mode 5 source modules ("MODUL5" and routines it calls) and "xxx_ISR5" for one example of a real-time, interrupt-driven structure.

Related Topics

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface File List for Hypersignal

TMS320C3x Source Code Modification Process for Hypersignal

5.5.2 TMS320C3x Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal TMS320C3x DSP Source Code Interface distribution diskette:

T30DEF.ASM	variable & constant definition module
T30BEG.ASM	initialization, including ISR vectors
ISR0.ASM	continuous A/D ISR for all boards
ISR1.ASM	continuous D/A ISR for all boards

ISR2.ASM	Digital Scope ISR for all boards
ISR5.ASM	Spectrum Analyzer ISR for all boards
ISR6.ASM	stimulus & response ISR for all boards
MODUL0.ASM	foreground A/D to disk processing (mode=0)
MODUL1.ASM	foreground D/A from disk processing (mode=1)
MODUL2.ASM	foreground Digital Oscilloscope processing (mode=2)
MODUL3.ASM	FFT file-based acceleration processing (mode=3)
MODUL4.ASM	iFFT file-based acceleration processing (mode=4)
MODUL5.ASM	foreground Spectrum Analyzer process (mode=5)
MODUL6.ASM	foreground stimulus & response process (mode=6)
MODUL7.ASM	convolution/correlation file-based acceleration (mode=7)
ARCTAN.ASM	arctangent routine
CONVLV.ASM	convolution/correlation routine
DIVF.ASM	floating-point divide routine
FFT.ASM	complex FFT/iFFT routine of variable length N ($N=2^{m}$ **m)
FFTOUT.ASM	post-FFT processing (mag, phase, etc.)
LOG.ASM	log approximation routine
MAG.ASM	magnitude routine (complex input)
MAX.ASM	find maximum-in-buffer routine
MIN.ASM	find minimum-in-buffer routine
PHASE.ASM	4-quadrant phase routine (–π to +π; –pi to +pi, complex input)
SINTAB.ASM	sine value lookup-table for FFT use
SPAOUT.ASM	post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation
SQRT.ASM	squareroot routine

WINDOW.ASM	apply pre-defined window (used with FFTs)
FILTER.ASM	filter control routine for real-time filtering
FIR.ASM	FIR filter routine
IIR2.ASM	IIR (cascade sections, modified direct form) filter routine
IIR3.ASM	IIR (cascade sections, balanced pole-zero structure) filter routine
XFER.ASM	complex-ratio routine used for transfer function
ASM.BAT	batch file used to assemble any source module: type "ASM filename" (no extension needed)
LNK.BAT	batch file used to link all source modules: type "LNK"
T30LINK.CMD	link control file (command file for linker)

The distribution diskette does not initially contain all .OBJ files necessary to produce a complete baseline, default executable code module when the T30LINK.CMD control file (above) is invoked with the Texas Instruments LNK30 utility. Each .ASM module above must be assembled first.

Related Topics

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface File List for Hypersignal

TMS320C3x Source Code Modification Process for Hypersignal

5.5.3 TMS320C3x Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of TMS32030.OUT before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to TMS320C3x source code, keeping in mind the necessary shared variables between Hypersignal and the TMS320C3x (the T30DEF.ASM module is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .OBJ (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that you may use in the form of:

```
ASM filename <car ret>
```

to assemble source code modules (entering an ".ASM" extension after "filename" is optional). This file executes a

asm30 -L %1

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the T30LINK.CMD control file (use an ASCII text-editor to update the T30LINK.CMD file).

3) Re-link all TMS320C3x modules using the Texas Instruments LNK30 linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .OBJ files. LNK.BAT executes a

lnk30 T30LINK.CMD

statement, where "T30LINK.CMD" is an ASCII linker-control file that specifies all .OBJ files to be linked by the LNK30.EXE program. Note that T30LINK.CMD places shared and local variables at 400H and 480H, respectively. All code has been tested using version 4.5 of the Texas Instruments TMS320C3x/C4x development tools.

4) Copy the resulting "TMS32030.OUT" file to the directory with Hypersignal software, run Hypersignal, and invoke the appropriate functions. Hypersignal reads the "COFF" file format directly, downloads all code and data, and begins execution at the reset vector (location zero).

Additional Information:

TMS320C3x Source Code Interface Modification Notes

Related Topics

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface File List for Hypersignal

TMS320C3x Source Code Modification Process for Hypersignal

TMS320C3x Source Code Interface Modification Notes

1) Avoid using the default ".TEXT", ".DATA", or ".BSS" sections in any new code you might add. For some reason, the Texas Instruments linker wants to put these sections in front of all named sections, unless you specifically tell it otherwise in the .CMD file. If one of these sections loads in front of the "INIT" section, which is supposed to load at zero because it contains the reset vector, your code is dead.

2) *Do not declare new variables before or in the "HOST_COMM" section (file T30DEF.ASM).* These are the host-DSP intercommunication memory areas that must remain unchanged to allow normal Hypersignal operation.

3) Note that the start of executable code used by Hypersignal is at 500H--if you declare enough variables in section "LOCAL" (file T30DEF.ASM) or immediately after it, you could overrun the start

address. If necessary put the variables somewhere else or change the address of "INIT" (both in T30BEG.ASM and T30LINK.CMD). Also note that analog and acceleration processing data buffers start at address 2000H. If you add enough code, you could overrun this address. If so, change the buffer address value "ANABFK" only--Hypersignal will base all other buffers on this value. *If you need to move any buffer addresses, please see the T30DEF.ASM file--it contains memory maps and numerous suggestions on changing default addresses.*

4) For ASPI Banshsee boards, any variables between 400H and 500H (normally includes all shared and local variables) are translated to dual-port memory area 800400H-8004FFH by the Hypersignal high-level downloader, and the "VARS" location is written as 80H. For the LSI board the 400H-4FFH area is translated to dual-port memory area 30400H, and the "VARS" location is written as 3. These relocations are done so that shared variables can be accessed by the host without putting the processor in hold.

5) For the LSI board, data buffers are also moved to the dual-port memory area (instead of at 2000H, in the "program" memory area). See the T30DEF.ASM module for details on the exact translation that occurs and under what circumstances.

6) The LNK.BAT file produces a "T30LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

7) If you need to use an external sampling clock, follow the basic guidelines below:

1. Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.
2. Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level

software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.

3. If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).

4. Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

SPECIAL NOTES FOR THE TEXAS INSTRUMENTS EVM BOARD AND DOMAIN TECHNOLOGIES DSPCARD

The Texas Instruments EVM board and Domain Technologies DSPCard are special cases. For both these boards, the .ASM files EVM_INIT, GET_COMM, RD_DATA, and WR_DATA can be assembled and linked (using the EVMLNK.BAT and DOMLNK.BAT files) to produce the small "host interface" executable programs "EVM_INIT.OUT" and "DOM_INIT.OUT". These files must be downloaded and run by the Texas Instruments EVMLOAD program or Domain Technologies LDR_C30 program, respectively, prior to running Hypersignal. You can use DOS commands of the following form:

```
EVMLOAD   EVM_INIT.OUT <car ret>
```

or

```
LDR_C30   DOM_INIT.OUT <car ret>
```

The EVMLOAD.EXE file should be included with your EVM board from Texas Instruments. *You must enter the ".OUT" extension when using the EVMLOAD program. Also, version 1.51 and above of EVMLOAD requires the two files "RESVCT.001" and "RESVCT.002" to be present in order to run--otherwise it spits up an "unable to*

initialize emulator" message. A default "EVM_INIT.OUT" file is included on Hypersignal distribution diskette #2. Once loaded, the host interface provides a mechanism for Hypersignal to download further code and real-time data, as necessary for standard operation.

The Domain LDR_C30.EXE program can be found on the Domain "Development System for Texas Instruments 320C3x" disk. A default DOM_INIT.OUT file is included on Hypersignal distribution diskette #2.

The reason for separate host-interface files (as opposed to including the host interface section inside the standard TMS32030.OUT file) is that Texas Instruments apparently regards the downloading of code directly through the Test Bus Controller (TBC) embedded-emulation interface on the EVM board as proprietary (even though we asked for this information, they declined to provide it). Obviously, the use of separate .OUT files for programs that must coexist (and execute) at the same time can create problems. Below is some discussion of this situation and things to watch out for should you desire to use the Hypersignal DSP Source Code Interface to change TMS320C3x source code for either EVM or DSPCard operation.

The EVM_INIT.OUT and DOM_INIT.OUT code files expect to start execution at 300H by the. The area from 300H to 400H should be represented by an "uninitialized section" (as it is in the default Hypersignal TMS32030.OUT file), preventing the Hypersignal COFF file loader (or loaders from other vendors) from overwriting the EVM or DSPCard host interfaces, once present in memory and running.

When Hypersignal puts the EVM board or DSPCard in "reset", what it is actually doing is changing the reset vector to point to an idle address, where the TMS320C30 or TMS320C31 spins while still allowing the interrupt-driven host interface to function. When Hypersignal "runs" the EVM board or DSPCard, the reset vector is changed again to allow normal or user-defined functions to execute (using the reset vector in the TMS32030.OUT file). Hypersignal never puts the EVM TMS320C30 or DSPCard TMS320C31 chips in hold, since to do so would not allow any data transfer while in the hold state.

Should you get in to the technical details of it, you will find that the Hypersignal version of the EVM host interface is much less complicated than the example interface provided by Texas Instruments. No elaborate command formatting or parameter area is

required. Since commands can be byte-wide, the Hypersignal interface simply sends the command type, instead of a "command start" bit followed by parameters and a "command end" bit. In the Hypersignal interface, the starting address is simply a pre-defined data transfer of one command type; other command types delineate the basic transfer mode (read 16-bit data, read 32-bit data, write 16-bit data, etc.). The count parameter has been eliminated completely since the EVM host interface always auto-increments its transfer address, and the host software always controls the number of transfers.

Related Topics

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface File List for Hypersignal

TMS320C3x Source Code Modification Process for Hypersignal

5.6 NEC 77220 Source Code Interface

Main Topics:

Related Topics:

77220 Source Code Interface Host PC Communication

77220 Source Code Interface Real-Time Processing Structure

77220 Source Code Interface Modification Notes

5.6.1 NEC 77220 Source Code Interface for Hypersignal

The Hypersignal 77220 DSP Source Code Interface distribution diskette contains all NEC 77220 source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic NEC 77220 source code includes analog I/O routines for the NEC DDK77220 board.

As is, the 77220 source code includes both library routines and also real-time interrupt driven "modes" of operation that use foreground processing and interrupt service routine processing to implement standard Hypersignal functions. These functions have been written in modular form, and support the basic "modes" of operation below:

77220 Module	Hypersignal Function	operating mode
MODUL0	Analog Conversion	mode=0
MODUL1	Analog Conversion, Playback from Waveform Display	mode=1
MODUL2	Digital Oscilloscope	mode=2
MODUL3	FFT file-based acceleration	mode=3
MODUL4	iFFT file-based acceleration	mode=4
MODUL5	Spectrum Analyzer	mode=5
MODUL6	Stimulus & Response operation	mode=6
MODUL7	Convolution/Correlation file-based acceleration	mode=7
MODULn	User-Defined function	mode=n

Related Topics

NEC 77220 Source Code Interface for Hypersignal

77220 Source Code Interface Host PC Communication

77220 Source Code Interface Real-Time Processing Structure

77220 Source Code Interface File List for Hypersignal

77220 Source Code Interface Host PC Communication

If the user wishes to alter the performance of the code, he/she must keep in mind the necessary links between Hypersignal software and the 77220 code. The two-way communication between the PC and 77220 established in the Hypersignal software must not be changed, as the high-level software expects to read/write certain memory locations. These memory locations are initialized by the high-level software before any code executes on the DSP board (some locations are actually read by the high-level software as well, allowing you to change such "constants" as buffer base addresses). *Do not insert any code of any type that will disturb these addresses*. The 77220 source code has been documented to aid the user in these areas.

Related Topics

NEC 77220 Source Code Interface for Hypersignal

77220 Source Code Interface Host PC Communication

77220 Source Code Interface Real-Time Processing Structure

77220 Source Code Interface File List for Hypersignal

77220 Source Code Modification Process for Hypersignal

77220 Source Code Interface Real-Time Processing Structure

The 77220 only has one external interrupt (except for NMI) which is currently used to implement a 'pseudo-hold' function to facilitate communication with the host PC (see 77220 Source Code Interface Modification Notes, below). Thus, for the DDK 77220 board implementation, all analog I/O is performed via polling of the onchip serial port status, requiring that all analog I/O data processing be done in the "foreground", on a buffer-loop basis. Since further processing on buffers must begin as soon as they are full, this means that if the processing to be done is significant (which it is for the Spectrum

Analyzer function), then the next data buffer won't be continuous, because no further samples can be acquired (without a sample interrupt) until the current buffer is completely processed.

Related Topics

NEC 77220 Source Code Interface for Hypersignal

77220 Source Code Interface Host PC Communication

77220 Source Code Interface Real-Time Processing Structure

77220 Source Code Interface File List for Hypersignal

77220 Source Code Modification Process for Hypersignal

5.6.2 77220 Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal 77220 DSP Source Code Interface distribution diskette:

N20DEF.SRC	variable & constant definition module
N20BEG.SRC	initialization, including ISR vectors
MODUL0.SRC	foreground A/D to disk processing (mode=0)
MODUL1.SRC	foreground D/A from disk processing (mode=1)
MODUL2.SRC	foreground Digital Oscilloscope processing (mode=2)
MODUL3.SRC	FFT file-based acceleration processing (mode=3)
MODUL4.SRC	iFFT file-based acceleration processing (mode=4)
MODUL5.SRC	foreground Spectrum Analyzer process (mode=5)
MODUL6.SRC	foreground stimulus & response processing (mode=6)
ARCTAN.SRC	arctangent routine
DIV.SRC	integer-divide routine
FFT.SRC	complex FFT/iFFT routine of variable length N (N=2**m)

FFTOUT.SRC	post-FFT processing (mag, phase, etc.)
LOG.SRC	log approximation routine
MAG.SRC	magnitude routine (complex input)
MAX.SRC	find maximum-in-buffer routine
MIN.SRC	find minimum-in-buffer routine
PHASE.SRC	4-quadrant phase routine (–π to +π; –pi to +pi, complex input)
SINTAB.SRC	sine value lookup-table for FFT use
SPAOUT.SRC	post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation
SQRT.SRC	squareroot routine
WINDOW.SRC	apply pre-defined window (used with FFTs)
FILTER.SRC	filter control routine for real-time filtering
FIR.SRC	FIR filter routine
IIR2.SRC	IIR (cascade sections, modified direct form) filter routine
IIR3.SRC	IIR (cascade sections, balanced pole-zero structure) filter routine
EXTRN.INC	include file containing reference to external variable definitions for use in other .SRC files (see N20DEF.SRC above)
SRC.BAT	batch file used to assemble any source module: type "SRC filename" (no extension needed)
ASM.BAT	batch file used to assemble any assembly module: type "ASM filename" (no extension needed)
LNK.BAT	batch file used to link all source modules: type "LNK"
N20LINK.CTL	link control file (command file for linker)

The distribution diskette does not initially contain all .REL files necessary to produce a complete baseline, default executable code module when the N20LINK.CTL control file (above) is invoked with

the NEC LK77230 utility. Each .SRC module above must be assembled first.

Related Topics

NEC 77220 Source Code Interface for Hypersignal

77220 Source Code Interface Host PC Communication

77220 Source Code Interface Real-Time Processing Structure

77220 Source Code Interface File List for Hypersignal

77220 Source Code Modification Process for Hypersignal

5.6.3 77220 Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of NEC77220.HEX before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to 77220 source code, keeping in mind the necessary shared variables between Hypersignal and the 77220 (the N20DEF.SRC module is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .OBJ (object) file is generated for each module. For convenience, a "SRC.BAT" batch file has been provided that you may use in the form of:

```
SRC filename <car ret>
```

to assemble source code modules (an ".SRC" extension after "filename" is optional). This file executes a

```
ra77230 nopre=%1 cp=20 object=%1 pri=%1.LST
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the N20LINK.CTL control file (use an ASCII text-editor to update the N20LINK.CTL file).

3) Re-link all 77220 modules using the NEC 77220 linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .LNK files. LNK.BAT executes the following statements:

```
lk77230 !N20LINK.CTL oc77230 NEC77220.LNK to NEC77220.TMP
hibyt220
NEC77220.TMP NEC77220.HEX del NEC77220.TMP
```

where "N20LINK.CTL" is an ASCII linker-control file that specifies all .LNK files to be linked by the LK77230.EXE program. Note that N20LINK.CTL blocks out (reserves) memory from 1000H to 13FFH for interrupt handling, shared and local variables, and initialization code. Thus, all relocatable code starts at 1400H. All code has been tested using version 2.0 of the NEC 77230/77220 development tools.

4) Copy the resulting "NEC77220.HEX" file to directory with Hypersignal software, run Hypersignal, and use appropriate functions. Hypersignal reads the "Intel hex" file format directly, downloads all code and data, and begins execution at the reset vector (location zero).

Additional Information:

77220 Source Code Interface Modification Notes

Related Topics

NEC 77220 Source Code Interface for Hypersignal

77220 Source Code Interface Host PC Communication

77220 Source Code Interface Real-Time Processing Structure

77220 Source Code Interface File List for Hypersignal

77220 Source Code Modification Process for Hypersignal

77220 Source Code Interface Modification Notes

1) Although hard-core NEC 77220 programmers might say that the 77220 assembly language (".ASM" files) is the only way to go, our

programmers, with a great deal of experience in programming many other DSP devices, felt that the "pre-assembly" language (".SRC" files) was much more like other DSP assembly languages and thus made creation of the 77220 source code more standard and easy-to-read. We have found the pre-assembler to do an excellent job in organizing code to take advantage of pipeline "delay tricks" in the 77220, as well as reliable in ensuring that either a NOP or non-related instruction is inserted when the pre-assembly source calls for use of a result or register too early. Since the pre-assembler can output its equivalent ".ASM" file at any time for manual optimization by the user, we feel that the development time saved by the pre-assembler more than justifies its use. Thus, all Hypersignal 77220 routines are ".SRC" files.

2) Note that the "N20LINK.CTL" file (command file for linker) blocks out fast instruction memory from 0 to 3FFH. This is because these areas are occupied by absolute sections: "RESET" at 0, "INT" (interrupt service routine) at 100H, "SHARED_VARS" and "LOCAL_VARS" at 200H and 280H, respectively, and "INIT" at 300H. Thus, all relocatable modules start at 400H, and are organized into successive chunks as long as necessary by the order in which they appear in the ".CTL" file.

3) *Do not declare new variables before or in the "SHARED_VARS" section (file N20DEF.SRC) starting at 200H in fast instruction memory*. These are the shared host-DSP intercommunication memory areas that must remain unchanged to allow normal Hypersignal operation. To add memory variables, we suggest adding them after existing variables in the "LOCAL_VARS" section (also file N20DEF.SRC) starting at 280H. Note that these sections reserve memory from 200H to 2FFH in fast instruction memory, and that there is code both before and after this section, so be careful of adding too many variables.

4) The LNK.BAT file produces a "N20LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for

individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

5) The "pseudo-hold" interrupt routine at 100H responds to a "HOLD" interrupt from the host PC, saves any of the current machine state that might be modified, and calls a routine in internal instruction EPROM that waits until the host PC says it is "free to go." This effectively puts the 77220 in a hold state, allowing the host PC to access the external data/instruction memory on the board. This mechanism allows the high-level Hypersignal software to operate the DDK 77220 board in a real-time mode, transferring analog I/O data continuously when necessary, and also allows the high-level software and the low-level 77220 code to operate asynchronously using "message passing" without the need to reset the 77220 or synchronize using I/O register flags.

6) If you need to use an external sampling clock, follow the basic guidelines below:

1. Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.
2. Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.
3. If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).
4. Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

NEC 77220 Source Code Interface for Hypersignal

77220 Source Code Interface Host PC Communication

77220 Source Code Interface Real-Time Processing Structure

77220 Source Code Interface File List for Hypersignal

77220 Source Code Modification Process for Hypersignal

5.7 Analog Devices ADSP-21xx Source Code Interface

Main Topics:

Related Topics:

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface Modification Notes

5.7.1 Analog Devices ADSP-21xx Source Code Interface for Hypersignal

The Hypersignal ADSP-21xx DSP Source Code Interface distribution diskette contains all Analog Devices ADSP-21xx source code for ADSP-2100, ADSP-2101 and ADSP-2115 boards, including linker

control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic ADSP-21xx source code includes analog I/O interrupt service routines for the LSI/Spectrum 2100 and 2101 system boards.

As is, the ADSP-21xx source code includes both library routines and also real-time interrupt driven "modes" of operation that use foreground processing and interrupt service routine processing to implement standard Hypersignal functions. These functions have been written in modular form, and support the basic "modes" of operation below:

ADSP-21xx Module	Hypersignal Function	operating mode
MODUL0	Analog Conversion	mode=0
MODUL1	Analog Conversion, Playback from Waveform Display	mode=1
MODUL2	Digital Oscilloscope	mode=2
MODUL3	FFT file-based acceleration	mode=3
MODUL4	iFFT file-based acceleration	mode=4
MODUL5	Spectrum Analyzer	mode=5
MODUL6	Stimulus & Response operation	mode=6
MODUL7	Convolution/Correlation file-based acceleration	mode=7
MODULn	User-Defined function	mode=n

Related Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface File List for Hypersignal

ADSP-21xx Source Code Modification Process for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on ADSP-21xx boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.7.7 above, ASDP-21xx Source Code Interface for Hypersignal) that they support, for example "ISR0". Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization in the initial setup code in "A21BEG", depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

Related Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface File List for Hypersignal

ADSP-21xx Source Code Modification Process for Hypersignal

ADSP-21xx Source Code Interface Host PC Communication

If the user wishes to alter the performance of the code, he/she must keep in mind the necessary links between Hypersignal software and ADSP-21xx code. The two-way communication between the PC and ADSP-21xx processor established in the Hypersignal software must not be changed, as the high-level software expects to read/write certain memory locations. These memory locations are initialized by the high-level software before any code executes on the DSP board (some locations are actually read by the high-level software as well, allowing you to change such "constants" as buffer base addresses). *Do not insert any code of any type before these addresses*. The ADSP-21xx source code has been documented to aid the user in these areas.

For all ADSP-21xx boards, you should avoid using the program memory areas between 1CH and 07FH, which is used by Hypersignal to store host-DSP interface variables and structures, and between 80H and 8FH, which is used by Hypersignal ADSP-21xx DSP code to store local variables. Note also that if Hypersignal ADSP-21xx DSP code is allowed to run, filter coefficients may be stored in onchip RAM at location 3C00H.

Related Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface File List for Hypersignal

ADSP-21xx Source Code Modification Process for Hypersignal

ADSP-21xx Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground. The dual-buffer communication between the host and DSP is fully synchronized using "interlock" buffer flags. Use mode 5 source modules ("MODUL5" and routines it calls) and "xxx_ISR5" for one example of a real-time, interrupt-driven structure.

Related Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface File List for Hypersignal

ADSP-21xx Source Code Modification Process for Hypersignal

5.7.2 ADSP-21xx Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal ADSP-21xx DSP Source Code distribution diskette:

A21DEF.DSP	variable & constant definition module
A21BEG.DSP	power-on/reset initialization
A21INT.DSP	interrupt and reset vector definition
ISR0.DSP	continuous A/D ISR for LSI/Spectrum system

	board
ISR1.DSP	continuous D/A ISR for LSI/Spectrum system board
ISR2.DSP	Digital Oscilloscope ISR for LSI/Spectrum system board
ISR5.DSP	Spectrum Analyzer ISR for LSI/Spectrum system board
ISR6.DSP	Stimulus & response ISR for LSI/Spectrum system board
MODUL0.DSP	foreground A/D to disk processing (mode=0)
MODUL1.DSP	foreground D/A from disk processing (mode=1)
MODUL2.DSP	foreground digital scope processing (mode=2)
MODUL3.DSP	FFT file-based acceleration processing (mode=3)
MODUL4.DSP	iFFT file-based acceleration processing (mode=4)
MODUL5.DSP	foreground Spectrum Analyzer process (mode=5)
MODUL6.DSP	foreground stimulus & response process (mode=6)
MODUL7.DSP	convolution/correlation file-based acceleration (mode=7)
ARCTAN.DSP	arctangent routine
BITREV.DSP	bit-reverse routine
CONVLV.DSP	convolution/correlation routine
DIVIDE.DSP	fractional and integer divide routines
FFT.DSP	complex FFT/iFFT routine of variable length N ($N=2^{m**m}$)
FFTOUT.DSP	post-FFT processing (mag, phase, etc.)
LOG.DSP	log approximation routine
MAG.DSP	magnitude routine (complex input)
MAX.DSP	find maximum-in-buffer routine
MIN.DSP	find minimum-in-buffer routine

PHASE.DSP	4-quadrant phase routine (–π to +π; –pi to +pi, complex input)
SINTAB.DSP	sine value lookup-table for FFT use
SPAOUT.DSP	post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation
SQRT.DSP	squareroot routine
WINDOW.DSP	apply pre-defined window (used with FFTs)
FILTER.DSP	filter control routine for real-time filtering
FIR.DSP	FIR filter routine
IIR2.DSP	IIR (cascade sections, modified direct form) filter routine
IIR3.DSP	IIR (cascade sections, balanced pole-zero structure) filter routine
XFER.DSP	complex-ratio routine used for transfer function
ASM.BAT	batch file used to assemble any source module: type "ASM filename" (no extension needed)
LNK.BAT	batch file used to link all source modules: type "LNK"
A21LINK.CTL	link control file (command file for linker)
HSMACRO.SYS	"system builder" file
HSMACRO.ACH	"system architecture" file

The distribution diskette does not initially contain all .OBJ, .CDE, and .INT files necessary to produce a complete baseline, default executable code module when the A21LINK.CTL control file (above) is invoked with the Analog Devices LD21 link utility. Each .DSP module above must be assembled first.

Related Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface File List for Hypersignal

ADSP-21xx Source Code Modification Process for Hypersignal

5.7.3 ADSP-21xx Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of ADSP21xx.EXE before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to ADSP-21xx source code, keeping in mind the necessary shared variables between Hypersignal and the ADSP-21xx (the A21DEF.DSP module is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .OBJ (object), .CDE, and .INT files are generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that you may use in the form of:

```
ASM filename <car ret>
```

to assemble source code modules (entering a ".DSP" extension after "filename" is optional). This file executes a

```
ASM21 -L %1
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the A21LINK.CTL control file (use an ASCII text-editor to update the A21LINK.CTL file).

3) Re-link all ADSP-21xx modules using the Analog Devices LD21 linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .OBJ, .CDE, and .INT files. LNK.BAT executes the following statements:

```
LD21 A21LINK.CTL -a HSMACRO -e AD21xx -g -x
DEL A21LINK.MAP
REN AD21xx.MAP A21LINK.MAP
```

where "A21LINK.CTL" is an ASCII linker-control file that specifies all .OBJ, .CDE, and .INT files to be linked by the LD21 link utility, and "A21LINK.MAP" is a map file showing variable and module addresses and lengths (see ADSP-21xx Source Code Interface Modification Notes, below). Note that A21LINK.CTL places shared and local variables at 001CH, forcing relocatable code to start after these declarations. All code has been tested using version 1.13 (11/21/90) of the Analog Devices ASM21 assembler, and version 1.12 (11/15/90) of the Analog Devices LD21 linker.

4) Copy the resulting "ADSP21xx.EXE" file to the directory with Hypersignal software, run Hypersignal, and invoke the appropriate functions. Hypersignal reads the Analog Devices .EXE file format directly, downloads all code and data, and begins execution at the reset vector (location zero).

Additional Information:

ADSP-21xx Source Code Interface Modification Notes

Related Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface File List for Hypersignal

ADSP-21xx Source Code Modification Process for Hypersignal

ADSP-21xx Source Code Interface Modification Notes

1) The basic LSI ADSP-2100 and ADSP-2101 boards support only 4kX24 of program memory. Thus, you must be careful when adding program memory variables or code so that overall code space is not exceeded. Program memory can be expanded to 16kX24.

2) Because of the 4kX24 minimum program memory size, the look-up table used by the Hypersignal FFT routine, which is stored in program memory, has been limited to a 1024-pt. FFT size. Because the FFT routine used requires the table to be in bit-reversed order, both cosine and sine values are required. Unfortunately, the ADSP-21xx bit-reversed addressing facility is not very flexible, not only placing restrictions on where such values are located in memory, but also creating the actual bit-reversed address only in hardware without making it available in software (via register). Should you desire to increase the FFT size, you must obtain expanded memory and also create a new bit-reversed look-up table.

3) In its factory configuration, the LSI ADSP-2101 board supports only 4kX16 of data memory. Data memory can be expanded, but not to 16kX16 because the 2101 only provides 14 external address lines, and in addition, uses the upper 4kX16 of data memory space for onchip internal RAM and other chip peripherals (serial ports, etc.). Thus, the maximum available data memory size for standard Hypersignal purposes is 12kX16.

4) If you need to use an external sampling clock, follow the basic guidelines below:

1. Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.
2. Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the

hardware, by directly changing the DSP source code as appropriate.

3. If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". The value is in Hz (Hertz); note that for ADSP-21xx boards, which use 16-bit memory accesses, the maximum value that can be represented is limited to 65535. This value is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).

4. Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

5) The LNK.BAT file produces a "A21LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

Related Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface File List for Hypersignal

ADSP-21xx Source Code Modification Process for Hypersignal

5.8 Motorola DSP96002 Source Code Interface

Main Topics:

Related Topics:

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface Modification Notes

5.8.1 Motorola DSP96002 Source Code Interface for Hypersignal

The Hypersignal DSP96002 DSP Source Code Interface distribution diskette contains all Motorola DSP96002 source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic DSP96002 source code includes analog I/O interrupt service routines for the LSI/Spectrum and Ariel DSP-96 boards.

As is, the DSP96002 source code is written in modular form, and supports the basic "modes" of operation below:

DSP96002 Module	Hypersignal Function	operating mode
MODUL0	Analog Conversion	mode=0
MODUL1	Analog Conversion, Playback from Waveform Display	mode=1
MODUL2	Digital Oscilloscope	mode=2

MODUL3	FFT file-based acceleration	mode=3
MODUL4	iFFT file-based acceleration	mode=4
MODUL5	Spectrum Analyzer	mode=5
MODUL6	Stimulus & Response operation	mode=6
MODUL7	Convolution/Correlation file-based acceleration	mode=7
MODULn	User-Defined function	mode=n

Related Topics

Motorola DSP96002 Source Code Interface for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface File List for Hypersignal

DSP96002 Source Code Modification Process for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the DSP96002 board and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.8.1 above, Motorola DSP96002 Source Code Interface for Hypersignal) that the routine supports, for example "ISR0" for operating mode zero. Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization in the "M96DEF" (non-executable definitions), "M96VEC", and

"M96BEG" modules, depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

Related Topics

Motorola DSP96002 Source Code Interface for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface File List for Hypersignal

DSP96002 Source Code Modification Process for Hypersignal

DSP96002 Source Code Interface Host PC Communication

If the user wishes to alter or add code functionality, or improve the functionality and/or performance of existing code, he/she must keep in mind the necessary intercommunication variables between Hypersignal software and DSP96002 code. The two-way communication between the host-PC and DSP96002 set up in the Hypersignal software must not be changed, as the software expects to read/write certain memory locations. These memory locations reside at 100H and are clearly commented inside M96DEF. *Do not insert any code of any type that would change the current addresses of these variables*. If necessary, you may add further communication variables for your own high-level code. The values of ANABFK, WNDBFK, RIBUFK, and MPBUFK (time data buffer, FFT window data buffer, real/imaginary data buffers, & magnitude/phase data buffers, respectively) can be changed if it becomes necessary to alter memory allocation in your DSP96002 code and communicate these changes to the host PC). The DSP96002 source code has been well-documented to aid the user in these areas.

For all DSP96002 boards, you should avoid using the program memory areas between 1CH and 03FH, which is used by Hypersignal to implement a host-interface, between 40H and 0FFH, which is used by Hypersignal to implement setup and initialization code necessary to

coordinate execution and DMA memory-mapped interfaces with the host, between 100H and 17FH, which is used by Hypersignal to store host-DSP interface variables and structures, and between 180H and 1FFH, which is used by Hypersignal DSP96002 DSP code to store local variables. Note that you can take advantage of the start up code located between by 40H and 0FFH by locating the start of your executable code at 400H, which is the initial jump taken by Hypersignal DSP96002 code after all setup and communication initialization is complete.

Related Topics

Motorola DSP96002 Source Code Interface for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface File List for Hypersignal

DSP96002 Source Code Modification Process for Hypersignal

DSP96002 Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts, although all ISRs save all registers temporarily used. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground.

Use mode 5 source modules ("MODUL5" and modules it calls), and "ISR5", for a good example of a real-time, interrupt-driven structure.

Related Topics

Motorola DSP96002 Source Code Interface for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface File List for Hypersignal

DSP96002 Source Code Modification Process for Hypersignal

5.8.2 DSP96002 Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal DSP96002 DSP Source Code Interface distribution diskette:

M96DEF.ASM	variable & constant definition module
M96VEC.ASM	beginning vectors, including ISR vector initialization, host-interface definition, and power-on/reset chip initialization
M96BEG.ASM	software initialization
ISR0.ASM	continuous A/D ISR for all boards
ISR1.ASM	continuous D/A ISR for all boards
ISR2.ASM	Digital Oscilloscope ISR for all boards
ISR5.ASM	Spectrum Analyzer ISR for all boards
ISR6.ASM	stimulus & response ISR for all boards
MODUL0.ASM	foreground A/D-to-disk processing (mode=0)
MODUL1.ASM	foreground D/A-from-disk processing (mode=1)
MODUL2.ASM	foreground Digital Oscilloscope processing (mode=2)
MODUL3.ASM	FFT file-based acceleration processing (mode=3)
MODUL4.ASM	iFFT file-based acceleration processing (mode=4)
MODUL5.ASM	foreground Spectrum Analyzer processing (mode=5)

MODUL6.ASM	foreground stimulus & response processing (mode=6)
MODUL7.ASM	convolution/correlation file-based acceleration processing (mode=7)
ARCTAN.ASM	arctangent routine
CONVLV.ASM	convolution/correlation routine
DIVIDE.ASM	fractional and integer divide routines
DMA_PREP.ASM	DMA transfer preparation routine (see DSP96002 Source Code Interface Modification Notes, below)
FFT.ASM	complex FFT/iFFT routine of variable length N ($N=2^{m}$**m)
FFTOUT.ASM	post-FFT processing (mag, phase, scaling)
FILTER.ASM	filter control routine for real-time filtering
FIR.ASM	FIR filter routine
IIR2.ASM	IIR (cascade sections, modified direct form) filter routine
IIR3.ASM	IIR (cascade sections, balanced pole-zero form) filter routine
LOG.ASM	log approximation routine
MAG.ASM	magnitude routine (complex input)
PHASE.ASM	4-quadrant phase routine (−π to +π; −pi to +pi, complex input)
SPAOUT.ASM	post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation
SQRT.ASM	squareroot routine
WINDOW.ASM	window routine: applies PC-downloaded window for Spectrum Analyzer and FFT
XFER.ASM	transfer function calculation (complex input)
XMAX.ASM	location of maximum value in X:buffer data

XMIN.ASM	location of minimum value in X:buffer data
YMAX.ASM	location of maximum value in Y:buffer data
XXMOVE.ASM YYMOVE.ASM XYMOVE.ASM YXMOVE.ASM LMOVE.ASM	move routines between X: and Y: memory
INTIOEQU.INC	include file containing equates for internal I/O addresses
EXTIOEQU.INC	include file containing equates for external I/O addresses
M96LINK.CTL	link control file (control file for linker)
ASM.BAT	batch file that can be used to assemble individual source code modules (use in form "ASM filename")
LNK.BAT	batch file that performs linking (use in form "LNK")

The distribution diskette does not initially contain all .CLN files necessary to produce a complete baseline, default executable code module when the M96LINK.CTL control file (above) is invoked with the Motorola DSPLNK utility. Each .ASM module above must be assembled first.

Related Topics

Motorola DSP96002 Source Code Interface for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface File List for Hypersignal

DSP96002 Source Code Modification Process for Hypersignal

5.8.3 DSP96002 Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of DSP96002.OUT before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to DSP96002 source code, keeping in mind the necessary links between Hypersignal and the DSP96002 (see discussion in DSP96002 Source Code Interface Host PC Communication, above, and M96DEF.ASM module, which is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .CLN (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that you may use in the form of:

```
ASM filename <car ret>
```

to assemble source code modules (entering an ".ASM" extension after "filename" is optional). This file executes a

```
asm96000 -B -L %1
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the M96LINK.CTL control file (use an ASCII text-editor to update the M96LINK.CTL file).

3) Re-link all DSP96002 modules using the Motorola DSPLNK linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .LNK files. LNK.BAT executes a

```
DSPLNK -BDSP96002.OUT -MM96LINK.MAP -FM96LINK.CTL
```

statement, where "M96LINK.CTL" is an ASCII linker-control file that specifies all .CLN files to be linked by the DSPLNK.EXE program. Note that re-locatable code is positioned in internal PRAM between

shared and local variables starting at 100H and software initialization code (M96BEG) starting at 400H (start of external PRAM), and afer M96BEG in external PRAM. This leaves significant space in internal PRAM for the linker to place interrupt service routines. You should not change absolute start of M96BEG at 400H unless you have a specific internal PRAM memory mapping strategy in mind; and you should not change the 100H address of shared variables for any reason, ever (as the host PC expects them there). All code has been tested using version 4.1 of ASM96000 and DSPLNK.

4) Copy the resulting "DSP96002.OUT" file to directory with Hypersignal, re-invoke the Hypersignal software, and run the appropriate function(s). Hypersignal reads the "COFF" file format directly, downloads all code and data, and begins execution at the reset vector (location zero).

Additional Information:

DSP96002 Source Code Interface Modification Notes

Related Topics

Motorola DSP96002 Source Code Interface for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface File List for Hypersignal

DSP96002 Source Code Modification Process for Hypersignal

DSP96002 Source Code Interface Modification Notes

1) If you need to change default buffer addresses, please consult the comments and suggestions in the M96DEF.ASM file.

2) The LNK.BAT file produces a "M96LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting

address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

3) The local variable "SEG" is used to store the current code segment page (upper 8 bits of linear address), and varies from board to board. Since virtually all DSP96002 program control instructions are not relocatable (instead using absolute addresses in the instruction sequence), all DSP96002 code must be dynamically relocated at run-time. After booting from internal PRAM, the DSP96002 code can then discover and set for future reference its code segment by examining instructions located in external PRAM.

4) In order to achieve the fastest possible means of transferring large amounts of continuous data in real-time between the host PC and the DSP96002, the Hypersignal-Macro implementation makes extensive use of the following combinations of transfer types, depending on the board in use:

a. onchip DMA between dual-port memory and other DSP96002 memory

b. onchip DMA between the host-interface and other DSP96002 memory

c. host-interface command-driven transfers

d. OnCE port driven transfers

It may be fair to say this is the most complex DSP chip interface supported by Hypersignal-Macro, of over 8 different DSP chips supported. It is definitely accurate to say that this is one of the most interesting and fascinating.

5) The LSI DSP96002 board uses IRQB for analog sample interrupts, and IRQC to initiate on-chip DMA controller data transfers. IRQA is unused. The Ariel DSP-96 board uses IRQA for analog sample interrupts, and IRQB to initiate on-chip DMA controller data transfers. IRQC is left available for use with the "heartbeat" clock interrupt on the DSP-96.

6) In no case does the Hypersignal-Macro DSP96002 source code use a dedicated register set for its host-interface definition. All registers used are saved temporarily on the stack, allowing use of a 'C' or other high-level compiler.

7) If you need to use an external sampling clock, follow the basic guidelines below:

1. Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.
2. Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.
3. If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).
4. Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

Motorola DSP96002 Source Code Interface for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface File List for Hypersignal

5.9 AT&T WE DSP3210 Source Code Interface

Main Topics:

Related Topics:

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface Modification Notes

5.9.1 AT&T WE DSP3210 Source Code Interface for Hypersignal

The Hypersignal DSP3210 DSP Source Code Interface distribution diskette contains all AT&T WE DSP3210 source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic DSP3210 source code includes interrupt service routines for the Ariel DSP-3210 board and CAC QUANTUMdsp board.

As is, the DSP3210 source code is written in modular form, and supports the basic "modes" of operation below:

DSP3210 Module	Hypersignal Function	operating mode
MODUL0	Analog Conversion	mode=0
MODUL1	Analog Conversion, Playback from Waveform Display	mode=1

MODUL2	Digital Oscilloscope	mode=2
MODUL3	FFT file-based acceleration	mode=3
MODUL4	iFFT file-based acceleration	mode=4
MODUL5	Spectrum Analyzer	mode=5
MODUL6	Stimulus & Response operation	mode=6
MODUL7	Convolution/Correlation file-based acceleration	mode=7
MODULn	User-Defined function	mode=n

Related Topics

AT&T WE DSP3210 Source Code Interface for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface File List for Hypersignal

DSP3210 Source Code Modification Process for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the DSP3210 boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.9.1 above, AT&T WE DSP3210 Source Code Interface for Hypersignal) that the routine supports, for example "ISR0" for operating mode zero. Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization

("itable" and PCW register) in RESET.S and W32BEG.S to locate the correct interrupt service routine for a particular function.

Related Topics

AT&T WE DSP3210 Source Code Interface for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface File List for Hypersignal

DSP3210 Source Code Modification Process for Hypersignal

DSP3210 Source Code Interface Host PC Communication

If the user wishes to alter or add code functionality, or improve the functionality and/or performance of existing code, he/she must keep in mind the necessary intercommunication variables between Hypersignal software and DSP3210 code. The two-way communication between the host-PC and DSP3210 device set up in the Hypersignal software must not be changed, as the software expects to read/write certain memory locations. These memory locations reside at locations "intercom" and "intermod" and are clearly commented inside W32DEF. *Do not insert any code of any type that would change the current addresses of these variables*. If necessary, you may add further communication variables for your own high-level code. The values of ANABFK, WNDBFK, RIBUFK, and MPBUFK (time data buffer, FFT window data buffer, real/imaginary data buffers, & magnitude/phase data buffers, respectively) can be changed if it becomes necessary to alter memory allocation in your DSP3210 code and communicate these changes to the host PC). The DSP3210 source code has been well-documented to aid the user in these areas.

For all DSP3210 boards, you should avoid using the area between 100H and 300H, which is used by Hypersignal to store host-DSP interface variables and structures (100H-1FFH) and local DSP variables (200H-3FFH).

Related Topics

AT&T WE DSP3210 Source Code Interface for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface File List for Hypersignal

DSP3210 Source Code Modification Process for Hypersignal

DSP3210 Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground.

Use mode 5 source modules ("MODUL5" and modules it calls), and "ISR5", for a good example of a real-time, interrupt-driven structure.

Related Topics

AT&T WE DSP3210 Source Code Interface for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface File List for Hypersignal

DSP3210 Source Code Modification Process for Hypersignal

5.9.2 WE DSP3210 Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal DSP3210 DSP Source Code Interface distribution diskette:

RESET.S	reset vector, wait-state, and "page register" initialization
W32DEF.S	variable & constant definition module
W32BEG.S	initialization, including ISR vectors
ISR0.S	continuous A/D ISR for all board
ISR1.S	continuous D/A ISR for all boards
ISR2.S	Digital Oscillscope ISR for all boards
ISR5.S	Spectrum Analyzer ISR for all boards
ISR6.S	Stimulus & response ISR for all boards
ISR6.S	Stimulus & response ISR for boards
ARI_TX.S	analog I/O output ISR for Ariel DSP-3210 only
MODUL0.S	foreground A/D to disk processing (mode=0)
MODUL1.S	foreground D/A from disk processing (mode=1)
MODUL2.S	foreground Digital Oscilloscope processing (mode=2)
MODUL3.S	FFT file-based acceleration processing (mode=3)
MODUL4.S	iFFT file-based acceleration processing (mode=4)
MODUL5.S	foreground Spectrum Analyzer process (mode=5)
MODUL6.S	foreground stim & response process (mode=6)
MODUL7.S	convolution/correlation file-based acceleration (mode=7)
ARCTAN.S	arctangent routine
CONVLV.S	convolution/correlation routine
DIV.S	divide routine
FFT.S	complex FFT/iFFT routine of variable length N

	(N=2^m**m)
FFTOUT.S	post-FFT processing (mag, phase, etc.)
FILTER.S	filter control routine for real-time filtering
FIR.S	FIR filter routine
IIR2.S	IIR (cascade sections, modified direct form) filter routine
IIR3.S	IIR (cascade sections, balanced pole-zero structure) filter routine
LOG10.S	log (base 10) approximation routine
MAXFIND.S	find maximum-in-buffer routine
MINFIND.S	find minimum-in-buffer routine
MAG.S	magnitude routine (complex input)
PHASE.S	4-quadrant phase routine (–π to +π; –pi to +pi, complex input)
SPAOUT.S	post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation
SQRT.S	squareroot routine
WINDOW.S	apply pre-defined window (used with FFTs)
XFER.S	complex-ratio routine used for transfer function
ASM.BAT	batch file used to assemble any source module: type "ASM filename" (no extension needed)
LNK.BAT	batch file used to link all source modules: type "LNK"
W32LINK.CTL	link control file (command file for linker)

The distribution diskette does not initially contain all .O files necessary to produce a complete baseline, default executable code module when the W32LINK.CTL control file (above) is invoked with the AT&T d32ld utility. Each .S module above must be assembled first.

Related Topics

AT&T WE DSP3210 Source Code Interface for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface File List for Hypersignal

DSP3210 Source Code Modification Process for Hypersignal

5.9.3 WE DSP3210 Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of DSP3210.OUT before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to DSP3210 source code, keeping in mind the necessary shared variable links between Hypersignal and the DSP3210 (see discussion in DSP3210 Source Code Interface Host PC Communication, above, and W32DEF.S module, which is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .O (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that you may use in the form of:

```
ASM filename <car ret>
```

to assemble source code modules (*do not enter a ".S" extension after "filename"*). This file executes a

```
d32as -b l -lo %1 %1.s
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the W32LINK.CTL control file (use an ASCII text-editor to update the W32LINK.CTL file).

3) Re-link all DSP3210 modules using the AT&T DL3D linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .O files. LNK.BAT executes the following statements:

```
D32LD -abs -map -o DSP3210.OUT W32LINK.CTL > W32LINK.MAP
@COPY W32LINK.MAP+EOF W32LINK.MAP
@D32NM -f DSP3210.OUT >> TEMP
@COPY W32LINK.MAP+TEMP W32LINK.MAP
@DEL TEMP
```

Additional Information:

DSP3210 Source Code Interface Modification Notes

Related Topics

AT&T WE DSP3210 Source Code Interface for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface File List for Hypersignal

DSP3210 Source Code Modification Process for Hypersignal

DSP3210 Source Code Interface Modification Notes

1) Hypersignal DSP3210 DSP source code uses r18 as a system stack pointer for both foreground and interrupt processing code. When storing CAU registers on the "stack", use the following sequence:

*r18++ = rN1
*r18++ = rN2
-
-

```
        -
*r18++ = rNN
```

When restoring CAU registers from the stack, use the following sequence:

```
rN1 = r18 – 4
rNN = *rN1--
        -
        -
        -
rN2 = *rN1--
rN1 = *rN1
r18 = r18 – K                /* K = 4 x (number of registers stored) */
```

There are three notable things about this process. First, the stack is always expected to increment and decrement in 4-byte chunks (always save/restore extended registers). Second, the stack is always expected to point to the *next available (free) location*. Third, when restoring, you can't simply do an r18e = r18 – 4 instruction and begin popping rXX = *r18-- instructions, because an interrupt might occur sometime after the r18 – 4 instruction and sometime before you finish popping; the interrupt routine, if it also used r18 as a stack pointer (which all types of routines should be able to do of a legitimate stack pointer), might then clobber a stored register (or more than one) that hadn't been popped yet.

2) Hypersignal DSP3210 source code is built with the "ASM" and "LNK" batch files, which invoke the AT&T "d32as" and "d32ld" assembler and linker utilities, respectively. By examining the ASM and LNK batch files, you should be able to construct an equivalent method of building the code using the AT&T "make" utility instead; however, producing a memory map file will be difficult. For the DSP Source Code Interface, obscure DOS commands have been used to create a map file in the LNK.BAT file, since the AT&T utilities will only print map output to the screen. The map file produced has a filename of "W32LINK.MAP"; you can use this file to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory,

problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

3) In addition to setting them directly in the DSP3210 source code, IOC register settings can be controlled from the high-level software in order to accommodate non-standard analog I/O connected to DSP3210 serial ports. Inside the Hypersignal-Macro configuration file there is a 32-bit integer IOC value for every DSP3210 board currently supported by Hypersignal-Macro--see the "CONFIG.DOC" file on Hypersignal distribution diskette #2 for a list of the sequence of variables in the configuration file. Depending on the board currently in use, one of these values is sent down to the DSP3210 board whenever a Hypersignal-Macro real-time function requiring analog I/O is invoked. The corresponding DSP source code variable is "IOC_reg", which is normally used directly to set the IOC register (unless changed by the user in the DSP source code).

4) If you need to use an external sampling clock, follow the basic guidelines below:

1. Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.
2. Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.
3. If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).

4. Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

AT&T WE DSP3210 Source Code Interface for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface File List for Hypersignal

DSP3210 Source Code Modification Process for Hypersignal

5.10 Texas Instruments TMS320C4x Source Code Interface

Main Topics:

Related Topics

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface Modification Notes

5.10.1 Texas Instruments TMS320C4x Source Code Interface for Hypersignal

The Hypersignal TMS320C4x DSP Source Code Interface distribution diskette contains all Texas Instruments TMS320C4x source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic TMS320C4x source code includes interrupt service routines for the Ariel DSP-C40 board.

As is, the TMS320C4x source code includes both library routines and also real-time interrupt driven "modes" of operation that use foreground processing and interrupt service routine processing to implement standard Hypersignal functions. These functions have been written in modular form, and support the basic "modes" of operation below:

TMS320C4x Module	Hypersignal Function	operating mode
MODUL0	Analog Conversion	mode=0
MODUL1	Analog Conversion, Playback from Waveform Display	mode=1
MODUL2	Digital Oscilloscope	mode=2
MODUL3	FFT file-based acceleration	mode=3
MODUL4	iFFT file-based acceleration	mode=4
MODUL5	Spectrum Analyzer	mode=5
MODUL6	Stimulus & Response operation	mode=6
MODUL7	Convolution/Correlation file-based acceleration	mode=7
MODULn	User-Defined function	mode=n

Related Topics

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface File List for Hypersignal

TMS320C4x Source Code Modification Process for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the TMS320C4x boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.10.1 above, Texas Instruments TMS320C4x Source Code Interface for Hypersignal) that they support, for example "ISR0". Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization in the initial setup code in "T40BEG", depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

Related Topics

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface File List for Hypersignal

TMS320C4x Source Code Interface Host PC Communication

If the user wishes to alter the performance of the code, he/she must keep in mind the necessary links between Hypersignal software and the TMS320C4x code. The two-way communication between the PC and TMS320C4x processor established in the Hypersignal software must not be changed, as the high-level software expects to read/write certain memory locations. These memory locations are initialized by the high-level software before any code executes on the DSP board (some locations are actually read by the high-level software as well, allowing you to change such "constants" as buffer base addresses). *Do not insert any code of any type before these addresses*. The TMS320C4x source code has been documented to aid the user in these areas.

For all TMS320C4x boards, you should avoid using the program memory areas between 0CH and 0C0H, which are reserved by Texas Instruments, between 0C0H and 400H, which is often used for debuggers and monitors, between 400H and 47FH, which is used by Hypersignal to store host-DSP interface variables and structures, and between 480H and 4FFH, which is used by Hypersignal TMS320C4x DSP code to store local variables.

Related Topics

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface File List for Hypersignal

TMS320C4x Source Code Modification Process for Hypersignal

TMS320C4x Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground. The dual-buffer communication between the host and DSP is fully synchronized using "interlock" buffer flags. Use mode 5 source modules ("MODUL5" and routines it calls) and "xxx_ISR5" for one example of a real-time, interrupt-driven structure.

Related Topics

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface File List for Hypersignal

TMS320C4x Source Code Modification Process for Hypersignal

5.10.2 TMS320C4x Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal TMS320C4x DSP Source Code Interface distribution diskette:

T40DEF.ASM	variable & constant definition module
T40BEG.ASM	initialization, including ISR vectors
ISR0.ASM	continuous A/D ISR for all boards
ISR1.ASM	continuous D/A ISR for all boards

ISR2.ASM	Digital Scope ISR for all boards
ISR5.ASM	Spectrum Analyzer ISR for all boards
ISR6.ASM	stimulus & response ISR for all boards
MODUL0.ASM	foreground A/D to disk processing (mode=0)
MODUL1.ASM	foreground D/A from disk processing (mode=1)
MODUL2.ASM	foreground Digital Oscilloscope processing (mode=2)
MODUL3.ASM	FFT file-based acceleration processing (mode=3)
MODUL4.ASM	iFFT file-based acceleration processing (mode=4)
MODUL5.ASM	foreground Spectrum Analyzer process (mode=5)
MODUL6.ASM	foreground stimulus & response process (mode=6)
MODUL7.ASM	convolution/correlation file-based acceleration (mode=7)
ARCTAN.ASM	arctangent routine
CONVLV.ASM	convolution/correlation routine
DIVF.ASM	floating-point divide routine
FFT.ASM	complex FFT/iFFT routine of variable length N ($N=2^{m}$ **m)
FFTOUT.ASM	post-FFT processing (mag, phase, etc.)
LOG.ASM	log approximation routine
MAG.ASM	magnitude routine (complex input)
MAX.ASM	find maximum-in-buffer routine
MIN.ASM	find minimum-in-buffer routine
PHASE.ASM	4-quadrant phase routine (–π to +π; –pi to +pi, complex input)
SINTAB.ASM	sine value lookup-table for FFT use
SPAOUT.ASM	post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation
SQRT.ASM	squareroot routine

WINDOW.ASM	apply pre-defined window (used with FFTs)
FILTER.ASM	filter control routine for real-time filtering
FIR.ASM	FIR filter routine
IIR2.ASM	IIR (cascade sections, modified direct form) filter routine
IIR3.ASM	IIR (cascade sections, balanced pole-zero structure) filter routine
XFER.ASM	complex-ratio routine used for transfer function
ASM.BAT	batch file used to assemble any source module: type "ASM filename" (no extension needed)
LNK.BAT	batch file used to link all source modules: type "LNK"
T40LINK.CMD	link control file (command file for linker)

The distribution diskette does not initially contain all .OBJ files necessary to produce a complete baseline, default executable code module when the T40LINK.CMD control file (above) is invoked with the Texas Instruments LNK40 utility. Each .ASM module above must be assembled first.

Related Topics

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface File List for Hypersignal

TMS320C4x Source Code Modification Process for Hypersignal

5.10.3 TMS320C4x Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of TMS32040.OUT before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to TMS320C4x source code, keeping in mind the necessary shared variables between Hypersignal and the TMS320C4x (the T40DEF.ASM module is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .OBJ (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that you may use in the form of:

```
ASM filename <car ret>
```

to assemble source code modules (entering an ".ASM" extension after "filename" is optional). This file executes a

```
asm30 -l -v40 %1
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the T40LINK.CMD control file (use an ASCII text-editor to update the T40LINK.CMD file).

3) Re-link all TMS320C4x modules using the Texas Instruments LNK30 linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .OBJ files. LNK.BAT executes a

```
lnk30 -v40 T40LINK.CMD
```

statement, where "T40LINK.CMD" is an ASCII linker-control file that specifies all .OBJ files to be linked by the LNK30.EXE program. Note that T40LINK.CMD places shared and local variables at 400H and 480H, respectively. All code has been tested using version 4.5 of the Texas Instruments TMS320C3x/4x development tools.

4) Copy the resulting "TMS32040.OUT" file to the directory with Hypersignal software, run Hypersignal, and invoke the appropriate functions. Hypersignal reads the "COFF" file format directly, downloads all code and data, and begins execution at the reset vector (location zero).

Additional Information:

TMS320C4x Source Code Interface Modification Notes

Related Topics

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface File List for Hypersignal

TMS320C4x Source Code Modification Process for Hypersignal

TMS320C4x Source Code Interface Modification Notes

1) Avoid using the default ".TEXT", ".DATA", or ".BSS" sections in any new code you might add. For some reason, the Texas Instruments linker wants to put these sections in front of all named sections, unless you specifically tell it otherwise in the .CMD file. If one of these sections loads in front of the "INIT" section, which is supposed to load at zero because it contains the reset vector, your code is dead.

2) *Do not declare new variables before or in the "HOST_COMM" section (file T40DEF.ASM).* These are the host-DSP intercommunication memory areas that must remain unchanged to allow normal Hypersignal operation.

3) Note that the start of executable code used by Hypersignal is at 500H--if you declare enough variables in section "LOCAL" (file T40DEF.ASM) or immediately after it, you could overrun the start

address. If necessary put the variables somewhere else or change the address of "INIT" (both in T40BEG.ASM and T40LINK.CMD). Also note that analog and acceleration processing data buffers start at address 2000H. If you add enough code, you could overrun this address. If so, change the buffer address value "ANABFK" only--Hypersignal will base all other buffers on this value. *If you need to move any buffer addresses, please see the T40DEF.ASM file--it contains memory maps and numerous suggestions on changing default addresses*.

4) The LNK.BAT file produces a "T40LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

5) If you need to use an external sampling clock, follow the basic guidelines below:

1. Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.
2. Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.
3. If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software,

based on what is entered in the current menu (SAMPLING FREQ field).

4. Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface File List for Hypersignal

TMS320C4x Source Code Modification Process for Hypersignal

5.11 Texas Instruments TMS320C5x Source Code Interface

Main Topics:

Related Topics

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface Modification Notes

5.11.1 Texas Instruments TMS320C5x Source Code Interface for Hypersignal

The Hypersignal TMS320C5x DSP Source Code Interface distribution diskette contains all TMS320C5x source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic TMS320C5x source code includes analog I/O interrupt service routines for the Innovative Integration PC50 board.

As is, the TMS320C5x source code is written in modular form, and supports the basic "modes" of operation below:

TMS320C5x Module	Hypersignal Functions	operating mode
MODUL0	Analog Conversion	mode=0
MODUL1	Analog Conversion, Playback from Waveform Display	mode=1
MODUL2	Digital Oscilloscope	mode=2
MODUL3	FFT file-based acceleration	mode=3
MODUL4	iFFT file-based acceleration	mode=4
MODUL5	Spectrum Analyzer	mode=5
MODUL6	Stimulus & Response operation	mode=6
MODUL7	Convolution/Correlation file-based acceleration	mode=7
MODULn	User-Defined function	mode=n

Related Topics

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface File List for Hypersignal

TMS320C5x Source Code Modification Process for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the TMS320C5x boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.11.1 above, Texas Instruments TMS320C5x Source Code Interface for Hypersignal) that they support, for example "ISR0". Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization in the initial setup code in "T50BEG", depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

Related Topics

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface File List for Hypersignal

TMS320C5x Source Code Modification Process for Hypersignal

TMS320C5x Source Code Interface Host PC Communication

If the user wishes to alter the performance of the code, he/she must keep in mind the necessary links between Hypersignal software and the TMS320C5x code. The two-way communication between the PC and TMS320C5x established in the Hypersignal software must not be changed, as the high-level host software expects to read/write certain memory locations. These memory locations are initialized by table reads from program memory inside module T50BEG, and are suffixed by 'ADD', as in MODADD, TRGADD, etc. *Do not insert any code of any type before these addresses that would cause them to move*. The TMS320C5x source code has been documented to aid the user in these areas.

For all TMS320C5x boards, you should avoid writing to program memory between 26H and 3FFH, which is typically left open for TMS320C5x debuggers, and 42DH and 4FFH, which is used by Hypersignal to store host-DSP communication variables and structures. Also, note that if Hypersignal TMS320C5x code is allowed to run, it stores local variables in onchip data memory locations 0-127, location 2FF, and 896 through 1023 (filter coefficients and delay elements).

Related Topics

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface File List for Hypersignal

TMS320C5x Source Code Modification Process for Hypersignal

TMS320C5x Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground. Use mode 5 source modules ("MODUL5" and modules it calls), and "ISR5" for one example of a real-time, interrupt-driven structure.

Related Topics

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface File List for Hypersignal

TMS320C5x Source Code Modification Process for Hypersignal

5.11.2 TMS320C5x Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal TMS320C5x DSP Source Code Interface distribution diskette:

T50DEF.ASM	variable & constant definition
T50BEG.ASM	beginning vectors, including ISR vectors, and including initialization, including PC-written variables
ISR0.ASM	continuous A/D ISR for all board
ISR1.ASM	continuous D/A ISR for all boards

ISR2.ASM	Digital Scope ISR for all boards.
ISR5.ASM	Spectrum Analyzer ISR for all boards.
ISR6.ASM	Stimulus & Response ISR for all boards.
MODUL0.ASM	foreground A/D-to-disk processing (mode=0)
MODUL1.ASM	foreground D/A-from-disk processing (mode=1)
MODUL2.ASM	foreground Digital Oscilloscope processing (mode=2)
MODUL3.ASM	FFT file-based acceleration processing (mode=3)
MODUL4.ASM	iFFT file-based acceleration processing (mode=4)
MODUL5.ASM	foreground Spectrum Analyzer process (mode=5)
MODUL6.ASM	foreground stimulus & response (mode=6)
MODUL7.ASM	convolution/correlation file-based acceleration processing (mode=7)
ARCTAN.ASM	arctangent routine
CONVLV.ASM	convolution/correlation routine
DIVIDE.ASM	fractional and integer divide routines
EXP2.ASM	exponentiation (base-2) routine (needed for fixed-point transfer function calculation)
FFT.ASM	complex FFT/iFFT routine of variable length N ($N=2^{m}$**m)
FFTOUT.ASM	post-FFT processing (mag, phase, scaling)
FILTER.ASM	invokes correct filter for current channel
FIR.ASM	FIR filter routine
IIR2.ASM	IIR filter cascade struc 2 (modified direct-form)
IIR3.ASM	IIR filter cascade struc 3 (balanced pole-zero)
LOG2.ASM	log2 approximation routine
MAG.ASM	magnitude routine
MAX.ASM	find maximum routine
MIN.ASM	find minimum routine

MOVE.ASM	move data
PHASE.ASM	4-quadrant phase routine (–π to +π; –pi to +pi, complex input)
SINTAB.ASM	sine value lookup-table for FFT use
SPAOUT.ASM	post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation
SQRT.ASM	squareroot routine
WINDOW.ASM	apply pre-defined window (used with FFTs)
XFER.ASM	complex-ratio routine used for transfer function
ASM.BAT	batch file used to assemble any source module: type "ASM filename" (no extension needed)
LNK.BAT	batch file used to link all source modules: type "LNK"
T50LINK.CMD	link control file (batch file for linker)

The distribution disk does not initially contain all .OBJ files necessary to produce a complete baseline, default executable code module when the T50LINK.CMD control file (above) is invoked with the Texas Instruments "dsplnk" utility. Each .ASM module above must be assembled first.

Related Topics

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface File List for Hypersignal

TMS320C5x Source Code Modification Process for Hypersignal

5.11.3 TMS320C5x Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of TMS32025.OUT before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to TMS320C5x source code, keeping in mind the necessary shared variables between Hypersignal and the TMS320C5x. The file T50DEF.ASM is well-documented in this area.

2) Assemble all modules changed and any new modules created (using Texas Instruments "COFF" assembler) to ensure that a new .OBJ (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that can be invoked as:

```
ASM filename <car ret>
```

to assemble source code modules (entering the ".ASM" extension after "filename" is optional). This file executes a

```
dspa -l -v50 -w %1
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the T50LINK.CMD control file (use an ASCII text-editor to update the T50LINK.CMD file).

3) Re-link all modules using Texas Instruments TMS320C5x "dsplnk" linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .OBJ files. LNK.BAT executes a

```
dsplnk T50LINK.CMD
```

statement, where "T50LINK.CMD" is an ASCII linker-command file that specifies all .OBJ files to be linked by the dsplnk.exe program, as well as specific information about memory addresses for some sections. Note that all executable code that is re-locatable starts at

400H. All code has been tested using version 6.40 of "dspa.exe" and "dsplnk.exe".

4) Copy the resulting "TMS32050.OUT" file to directory with Hypersignal software, run Hypersignal, and invoke the appropriate functions. Hypersignal reads the "COFF" file format directly, downloads all code and data, and begins execution at the reset vector (location zero).

Additional Information:

TMS320C5x Source Code Interface Modification Notes

Related Topics

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface File List for Hypersignal

TMS320C5x Source Code Modification Process for Hypersignal

TMS320C5x Source Code Interface Modification Notes

1) If you need to change default buffer addresses, please consult the comments and suggestions in the T50DEF.ASM file.

2) The LNK.BAT file produces a "T50LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching

for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

3) The T50LINK.CMD file creates an uninitialized "hole" from 26H through 3FFH that can be used for monitor and debugger programs. Because this section is uninitialized, the Hypersignal COFF loader will not write anything to this area.

4) If you need to use an external sampling clock, follow the basic guidelines below:

1. Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.
2. Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.
3. If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". The value is in Hz (Hertz); note that for TMS320C5x boards, which use 16-bit wide memory accesses, the maximum value that can be represented is limited to 65535. This value is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).
4. Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface File List for Hypersignal

TMS320C5x Source Code Modification Process for Hypersignal

Notes: